VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

BEZDRÁTOVĚ ŘÍZENÝ ZDROJ SVĚTLA

LIGHT SOURE WITH WIRELESS CONTROL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **VÁCLAV MARTINKA**
AUTHOR

VEDOUCÍ PRÁCE — **Prof. Dr. Ing. PAVEL ZEMČÍK**
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Martinka Václav**
Obor: Informační technologie
Téma: **Bezdrátově řízený zdroj světla**
Light Soure with Wireless Control
Kategorie: Uživatelská rozhraní

Pokyny:

1. Prostudujte způsob řízení světel počítačem se zaměřením na embedded systémy, bezdrátovou komunikaci a LED světelné zdroje.
2. Navrhněte zdroj světla s LED řízený embedded systémem a způsob jeho bezdrátového ovládání.
3. Popište a diskutujte možnosti navrženého řešení a popište předpokládané vlastnosti.
4. Navržený systém implementujte a demonstrujte jeho funkčnost.
5. Diskutujte dosažené výsledky a možnosti pokračování práce.

Literatura:

- Dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3 zadání

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zemčík Pavel, prof. Dr. Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Cílem této práce je navrhnout a implementovat zdroj světla bezdrátově ovládaný libovolným zařízením skrze Wi-Fi síť. Práce popisuje výběr komponent a jejich zapojení. Dále se věnuje knihovně pro Wi-Fi modul, způsobu komunikace po síti a návrhu uživatelského rozhraní. Výstupem práce je funkční prototyp umístěný na desce plošných spojů.

Abstract

The aim of this work is to design and implement a lighting which can be controlled wirelessy by any user device with Wi-Fi. The bachelor thesis describes the selection of components and their connection. Also describes a library for a Wi-Fi module, a way how to communicate over the network and design a user interface. The result of this thesis is a functional prototype placed on the printed circuit board.

Klíčová slova

LED osvětlení, bezdrátové ovládání, řízení mikropočítačem, Wi-Fi modul ESP8266, Arduino

Keywords

LED lighting, wireless control, microcomputer control, Wi-FI module ESP8266, Arduino

Citace

MARTINKA, Václav. *Bezdrátově řízený zdroj světla.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Prof. Dr. Ing. Pavel Zemčík

Bezdrátově řízený zdroj světla

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Prof. Dr. Ing. Pavla Zemčíka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Václav Martinka
13. května 2018

Poděkování

Děkuji mému vedoucímu Prof. Dr. Ing. Pavlu Zemčíkovi za odpornou pomoc při vypracování bakalářské práce.

Obsah

Kapitola 1

Úvod

V posledních letech došlo k masivnímu rozvoji v oblasti elektroniky. Zařízení jsou menší, levnější a výkonnější. Ve většině domácností tak najdeme počítač, notebook či chytrý telefon. Další rychle rozvíjející se technologií je LED osvětlení. Dříve byly *LEDky* drahou záležitostí s nízkým výkonem neschopné svítit přirozeným světlem, což už dávno neplatí. Naopak se čím dál tím více prosazují jako hlavní prvek osvětlení interiéru i exteriéru.

A právě kombinace těchto dvou technologií je tématem této bakalářské práce. Tedy konkrétně možnost pomocí počítače či telefonu ovládat domácí osvětlení postavené na principu LED. Elektronika a s tím související vestavěné systémy mě vždy zajímaly. To stejné lze říci i o LED osvětlení, tudíž mi toto téma bylo od začátku velmi blízké. Rád jsem proto využil příležitosti navrhnout řešení přesně podle mých představ.

Cílem práce bylo vyvinut takové zařízení, které by umožňovalo integraci se současným osvětlením. Bylo by multiplatformní s otevřeným komunikačním protokolem. Snadno postavitelné z běžně dostupných součástek a samozřejmě co nejlevnější.

V následující kapitole se věnuji současným technologiím jako jsou různé typy osvětlení a způsobům jejich ovládání. Dále zmiňuji princip činnosti mikrokontroléru, fungování počítačových sítí a webových technologií, jelikož se jedná o základní prvky potřebné k vytvoření zmíněného zařízení. Třetí kapitola hodnotí současná řešení a na základě toho upřesňuje zadání a stanovuje dílčí cíle. Popis samotné práce naleznete v kapitole číslo 4, kde se zaměřuji na návrh zapojení, komunikaci mezi uživatelem a vznikajícím zařízením, ale např. i bezpečnostní rizika. Poslední kapitola v několika odstavcích shrnuje, jak bylo dosaženo cíle a co by bylo možné dále vylepšit.

Kapitola 2

Současné technologie

Tato kapitola si klade za cíl seznámit čtenáře s technologiemi, které lze využít při stavbě bezdrátově ovládaného osvětlení. Těmi nejsou jen samotná svítidla jako žárovky, či jedno i vícebarevné LED. Naopak sem lze zařadit také mikrokontrolér jakožto hlavní ovládací prvek zařízení či Wi-Fi modul pro samotnou bezdrátovou komunikaci. S tím úzce souvisí i fungování počítačových sítí a webových technologií obecně. Níže uvádím pouze informace bezprostředně nutné k pochopení práce, nejedná se o úplný popis všech existujících technologií a postupů.

2.1 Světelné zdroje – žárovky, zářivky a LED

S ohledem na zaměření práce pouze zmíním v domácnosti běžně používané světelné zdroje jakými jsou žárovky, zářivky a LED. Jiné typy svítidel (např. pouliční osvětlení) jsou často technologicky blízké výše zmíněným typům a jejich ovládání je tudíž velmi podobné, ne-li stejné.

2.1.1 Žárovka

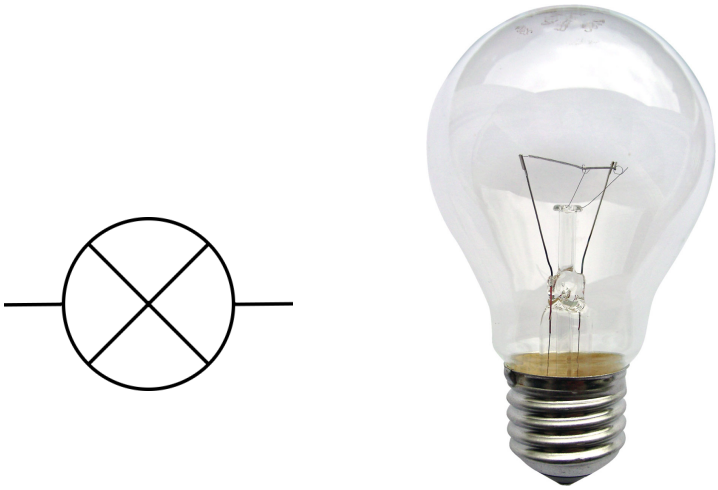
Žárovky patří mezi nejstarší a nejlevnější zdroje světla. Ke svícení využívají zahřívání vlákna protékajícím proudem. To při vysokých teplotách (okolo 2 500 °C) začne vydávat i viditelné světlo, ovšem většina záření je vyzařována v infračervené oblasti. Toto vlákno bývá obvykle vyrobeno z wolframu a umístěno do skleněné baňky obsahující buď vakuum nebo směs inertních plynů (zpravidla krypton nebo xenon s 10% příměsí dusíku). [22]

Dnes se od žárovek postupně upouští, jelikož jejich měrný světelný výkon je poměrně nízký, pohybuje se mezi 5 – 35 lm/W v závislosti na celkovém výkonu a konstrukci. [22] V domácnosti běžně používané 60W žárovky vykazují cca 15 lm/W. [2]

Speciálním typem jsou halogenové žárovky, které jsou konstrukčně velmi podobné. Rozdíl oproti klasickým žárovkám je v použitém plynu uvnitř baňky, který tvoří halogenidy, a v mnohem vyšších pracovních teplotách. Díky tomu tyto žárovky vydávají bělejší a intenzivnější světlo. [22].

2.1.2 Zářivka

Zářivkou je nejčastěji myšlena podlouhlá uzavřená trubice s dvojicí elektrod na každém konci naplněná směsicí plynů (konkrétně rtutí a argonem) o nízkém tlaku. Světlo zde vzniká jako výboj v plynu, pro jehož zažehnutí je třeba vysokonapěťového impulsu, tudíž trubice



Obrázek 2.1: Schématická značka a fotka běžné žárovky (*KMJ, CC-BY-SA-3.0*)

musí být doplněna o další elektroniku. Při výboji vzniká UV záření, proto tělo trubice zevnitř pokrývá vrstva tzv. luminoforu[1], který ovlivňuje výslednou vyzařovanou barvu. [22]

Zářivky se spíše než v domácnosti nachází ve veřejných budovách (školy, úřady) z důvodu vyšších výkonů oproti žárovkám (50 – 70 lm/W) a delší životnosti. To je vykoupeno většími rozměry a většinou nepřirozeným studeným světlem. [2]

Pro použití v domácnostech vznikly tzv. *kompaktní zářivky* (často nesprávně označované jako úsporné žárovky). Ty v těle obsahují veškerou elektroniku (proto je možné ji přímo připojit na zdroj napětí) a trubice zpravidla není rovná, nýbrž nějakým způsobem zkroucená z důvodu kompaktnosti. [22] Jejich výkon je sice menší (cca o 5 lm/W), ale stále více jak dvojnásobný oproti žárovkám. [2]

(a) Detail zářivkové trubice (b) Kompaktní zářivka

Obrázek 2.2: Příklad různých zářivek

[1] Luminofor je látka schopná pohlcovat energii a následně ji vyzařovat ve formě světla (tzv. luminiscence).

2.1.3 LED

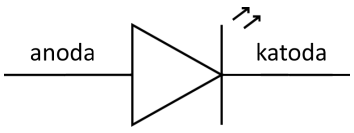
LED z anglického *Light-Emitting Diode* (česky *dioda emitující světlo*), je polovodičová součástka obsahující jeden PN přechod, která při průchodu proudu v průchozím směru vyzařuje světlo v úzkém barevném spektru. [17] Velmi často se také vyskytuje nevhodné označení LED dioda, zde ovšem dochází k *pleonasmu*[2], jelikož zkratka LED již slovo dioda obsahuje.

Vyzařovanou barvu určuje chemické složení použitých polovodičů. Protože bílé světlo je složeno kombinací více barev, není možné ho přímo vytvořit pomocí jedné diody. Buď se musí použít více čipů (využívá se RGB model, viz 2.3.2) nebo doplnit pouzdro diody podobně jako u zářivky o luminofor. [17]

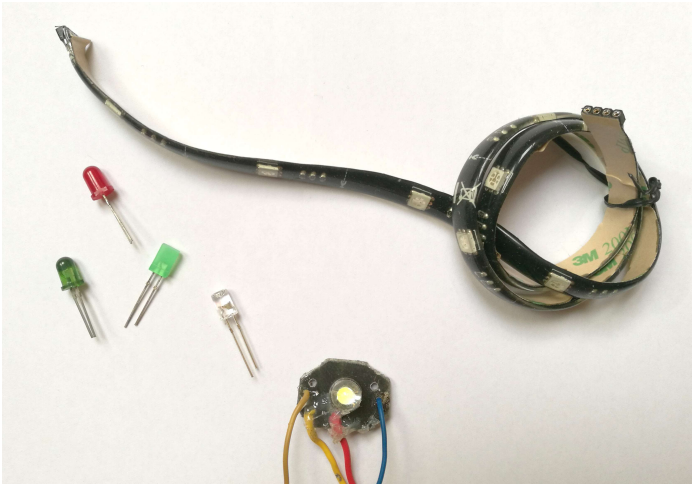
V domácnosti se nejčastěji vyskytují v podobě *LED žárovek*, které obsahují větší množství čipů umístěných tak, aby světlo vyzařovali do všech úhlů, a veškerou potřebnou elektroniku. Dalším možným provedením jsou *LED pásky*, což jsou pružné samolepící pásky členěné na jednotlivé moduly (díky tomu ho lze dle potřeby krátit či spojovat) obsahující několik (většinou 1 nebo 3) výkonných LED na jednom modulu. Mohou být i vícebarevné (nejčastěji RGB), pak lze za pomoci řadiče reprodukovat různé barvy. [22]

Oproti předchozím zdrojům světla je použití LED o něco složitelnější. K napájení lze použít pouze stejnosměrné napětí. Jsou náchylné na vyšší proudy a teploty, výkonné LED se tudíž neobejdou bez pasivního chladiče. Tyto negativa se za uživatele snaží vyřešit výrobce právě produkcí již hotových řešení jako jsou výše zmíněné LED žárovky.

LED nabízí vysoký měrný světelný výkon – pohybuje se od 70 lm/W až po 140 lm/W a díky vývoji se neustále zvyšuje [2]) – a dlouhou životnost. S ohledem na tyto skutečnosti tak v poslední době nachází stále větší uplatnění. A to nejen v rámci domácností, ale i v ručních svítilnách, pouličním osvětlení nebo třeba v automobilech. Dalším benefitem je snadné řízení jasu pomocí PWM, viz podkapitola 2.5. [22]



(a) Schématická značka LED



(b) Ukázka různých LED: 4 indikační (*vlevo*), 1 výkonná (*dole*), RGB pásek (*vpravo*)

Obrázek 2.3: LED

[2] Nadbytečné nahromadění významově blízkých výrazů.

2.2 Základní elektronické součástky a některá jejich zapojení

Každé složitější elektronické zařízení je složené z jednodušších celků. Tyto celky se přitom často opakují nejen v rámci konkrétního zapojení, ale i napříč různými zapojeními. Z toho důvodu vznikly integrované obvody, což jsou součástky, které v sobě obsahují často používaná zapojení. Ale i samotné integrované obvody jsou tvořeny základními elektronickými součástkami. Ty nejběžnější z nich zmíním v následujících odstavcích včetně několika velmi jednoduchých zapojení.

2.2.1 Rezistor

Často též nesprávně nazývaný *odpor* je pasivní elektronická součástka, používaná snad ve všech elektrických obvodech. V ideálním případě má v zapojení jedinou vlastnost – elektrický odpor, tedy přeměnou elektrické energie na teplo (rezistor elektrickou energii spotřebuje). V reálném obvodu se mimo jiné projevuje také nechtěnou indukčností, kapacitou a závislostí na okolní teplotě. Dále produkuje tepelné ztráty, které jsou při vyšších výkonech poměrně výrazné a je nutné ho chladit. [15]

(a) Evropský standard [8] (b) Americký standard [8]

Obrázek 2.4: Schématická značka pro rezistor

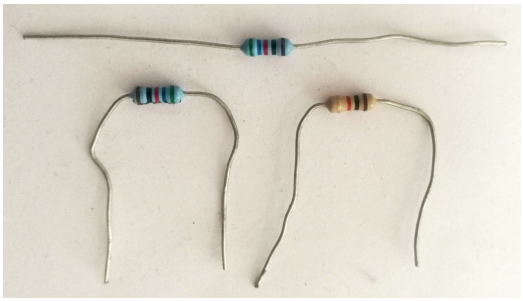
Rezistor se využívá k omezení proudu či úbytku napětí v obvodu. Tento vztah popisuje Ohmův zákon $U = R \cdot I$, kde R je odpor rezistoru ovlivňující proud (I) nebo napětí (U). Jeho schématická značka není celosvětově sjednocena, na obrázku 2.4 je vidět rozdíl mezi evropskou a americkou normou. Protože některý software pro kreslení schémat tyto rozdíly nezohledňuje, tak je možné i v našich končinách narazit na americkou variantu a naopak. [15]

Při výrobě se používá několik různých výrobních postupů, které mimo jiné určují výslednou cenu, rozměry, maximální výkon nebo toleranci (od 10 % až po 0,1 %). Dříve se hodnota rezistoru tiskla přímo na jeho tělo, později se přešlo ke značení pomocí barevných proužků, které je odolnější vůči poškození a je snadněji čitelné pro roboty. Ukázku běžně dostupných 0,6W rezistorů lze vidět na obrázku 2.5(a). Na sousedním obrázku 2.5(b) je znázorněna volt-ampérová charakteristika několika různých hodnot rezistorů, na které lze pozorovat lineární závislost napětí na proudu, která je přímo úměrná odporu konkrétního rezistoru. [15]

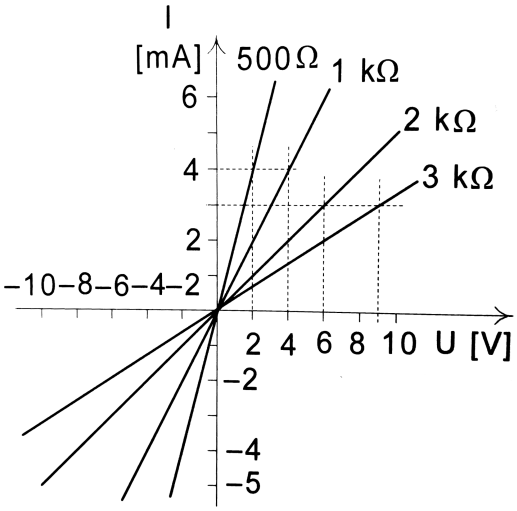
2.2.2 Dioda

Dioda je elektronická součástka schopná propouštět elektrický proud pouze jedním směrem. Dříve se tohoto efektu dosahovalo složitě za pomoci vakua (elektronky), dnes se využívají polovodiče, které jsou menší, účinější a mají nižší spotřebu. Tyto diody se pak nazývají *polovodičové* a jsou tvořeny jedním PN přechodem. [16]

Schématická značka diody je znázorněna na obrázku 2.6, jednotlivé vývody se nazývají anoda a katoda. Je-li na anodu přivedeno kladnější napětí než na katodu, je dioda zapojena v tzv. *propustném směru* (tento směr znázorňuje šipka na schématické značce). Opačné zapojení se nazývá *závěrný směr*. [16]



(a) Příklad běžně používaných rezistorů



(b) Volt-ampérová charakteristika rezistoru [15]

Obrázek 2.5: Ukázka rezistoru a jeho VA charakteristika

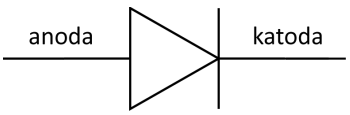
Propustný směr v rozsahu 0 – *prahové napětí* (pro běžnou křemíkovou diodu je to asi 0,6 V) nepropouští žádný proud – dioda se chová téměř jako izolant. Po překonání prahového napětí se z diody naopak stává velmi dobrý vodič s nízkým odporem a tak dohází k prudkému růstu proudu společně s rostoucím napětím. Není-li nějak omezen, dojde k tepelné destrukci diody. Běžné diody jsou stavěny na proudy v řádu jednotek ampér, u výkonných to mohou být i stovky. [16]

Závěrný směr omezuje protékající proud na minimum (mikroampéry) bez ohledu na napětí. Přesněji řečeno až do dosažení *závěrného napětí*, kdy dochází k proražení (zpravidla nezvratnému) PN přechodu a z diody se stává vodič. [16]

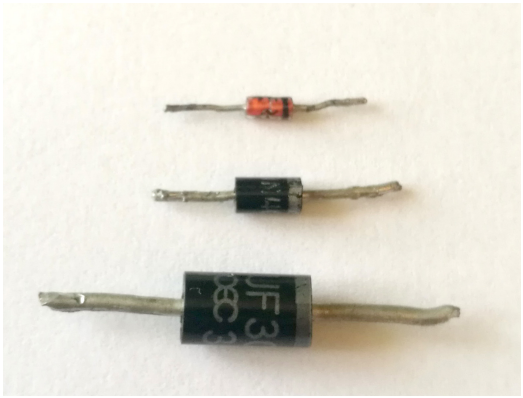
Úbytek napětí

Při průchodu proudu diodou (PN přechodem obecně) dochází k tzv. *úbytku napětí* o určitou hodnotu, která je téměř konstantní, tedy se se změnou proudu nemění. U dnešních polovodičových diod je to cca. 0,7 V. Tento úbytek se také značí jako V_F (z anglického *Voltage Forward*). [28]

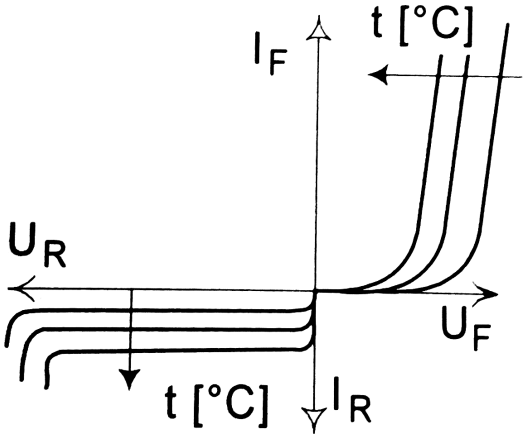
Diody nachází využití jako usměrňovače střídavého proudu, popř. jako ochrana před přepólováním. Kromě klasické usměrňovaní diody existují i další, např. Zenerova dioda, kterou lze využít ke stabilizaci napětí. Graf 2.7(b) znázorňuje volt-ampérovou charakteristiku diody a její závislost na teplotě. Je dobré si povšimnout skutečnosti, že s rostoucí teplotou se snižuje prahové napětí a naopak roste proud. Proto je nutné diody (a polovodiče obecně) chladit, jinak může dojít k jejich přetížení a destrukci.



Obrázek 2.6: Schématická značka



(a) Ukázka několika diod



(b) Volt-ampérová charakteristika diody [16]

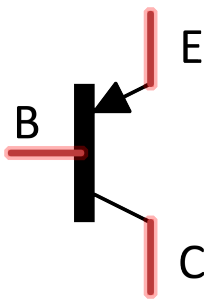
Obrázek 2.7: Ukázka diody a její VA charakteristika

2.2.3 Tranzistor

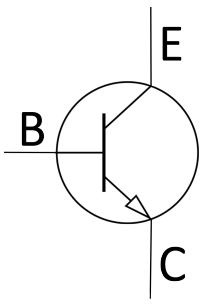
Objev tranzistoru se zasloužil o rozvoj a miniaturizaci elektroniky. Existují tři typy tranzistorů: *bipolární*, *unipolární* a *kombinované*. Dále se budu věnovat výhradně bipolárním tranzistorům, ale princip fungování ostatních typů je s drobnými odchylkami stejný. Zatímco dioda je tvořena jedním PN přechodem, tak tranzistor je tvořen dvěma přechody. Podle uspořádání těchto přechodů se tranzistory dělí na NPN a PNP. [16]

Jeho základní schopností je zesilovat proud, to znamená pomocí malé změny proudu na vstupu vyvolat vysoké změny proudu na výstupu. Díky tomu nalezneme tranzistory ve všech zapojeních, kde je potřeba pomocí digitálních signálů ovládat proudově náročná zařízení – používají se jako *budiče*. [28]

Uspořádání tranzistoru (to, zda se jedná o NPN nebo PNP) určuje pouze jeho polaritu, jinak se oba chovají stejně. Schématickou značku najdeme na obrázku 2.8. Je zde znázorněn nejen rozdíl mezi značkou pro PNP a NPN tranzistor (liší se ve směru šipky), ale také varianta s kružnicí okolo tranzistoru a bez ní. [28] Dále je vhodné si povšimnout, že tranzistor má tři elektrody: B – báze, E – emitor, C (K) – kolektor. Báze slouží k řízení proudu mezi emitorem a kolektorem. Tedy je to právě malý *bázový proud* I_B, který je tranzistorem zesílen. [16]



(a) PNP



(b) NPN

Obrázek 2.8: Schématická značka tranzistoru

2.2.4 Dělič napětí

Dělič napětí slouží k rozdělení napětí v určitém poměru za pomoci Ohmova zákona a sériového zapojení dvou a více rezistorů. Příklad takového zapojení znázorňuje obrázek 2.9. Výstupní napětí na pinu OUT je rovno napětí na rezistoru R_2 (měřeno vůči zemi), takže platí $U_{OUT} = U_2$. Z Ohmova zákona lze odvodit vzorce pro výpočet napětí na jednotlivých rezistorech

$$U_1 = R_1 \cdot I_1$$

$$U_2 = R_2 \cdot I_2$$

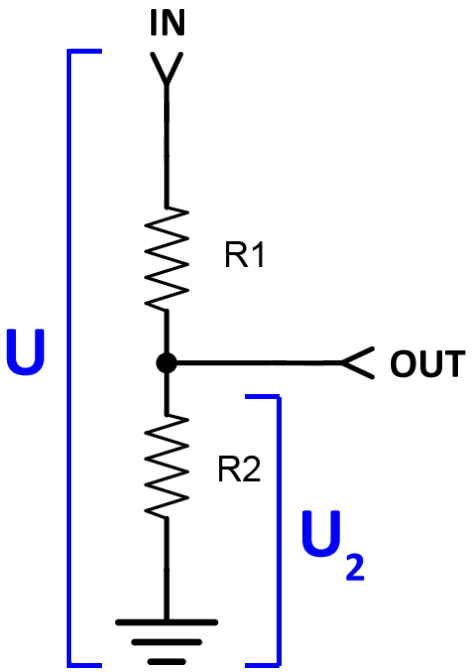
Protože se jedná o sériové zapojení, tak je proud protékající rezistory stejný a platí

$$I_1 = I_2 = I$$

Tyto tři rovnice udávají výsledný vztah

$$\frac{U_1}{U_2} = \frac{R_1}{R_2}$$

což znamená, že poměr napětí na rezistorech je přímo úměrný poměru odporu jednotlivých rezistorů. [28]



Obrázek 2.9: Dělič napětí

Dělič napětí lze využít například při měření, kdy je omezený vstupní rozsah měřícího přístroje na 0 – 5 V a výstup ze sondy je v rozsahu 0 – 50 V. Při výpočtu konkrétního rezistoru je nutné nejdříve určit poměr, kterým bude rozděleno napětí (dále značen písmenem x). Tudíž musí platit následující vztah

$$U_{OUT} = \frac{U_{IN}}{x}$$

Z předchozí rovnice vyplývá, že poměr napětí je přímo úměrný poměru odporů. Dosazením tak lze snadno získat poměr $R_1 : R_2$.

$$\begin{aligned} U_{OUT} &= \frac{U_{IN}}{x} \\ R_2 &= \frac{R}{x} \\ x \cdot R_2 &= R_1 + R_2 \\ x \cdot R_2 - R_2 &= R_1 \\ R_2 \cdot (x-1) &= R_1 \\ R_2 &= \frac{R_1}{x-1} \end{aligned}$$

Vzorec neurčuje konkrétní hodnoty rezistorů, pouze jejich poměr. Většinou se používají běžně dostupné rezistory v řádech kilo-ohmů, přičemž je nutné vzít v potaz určitou nepřesnost při jejich výrobě. Běžné rezistory mají toleranci 5 %, v případě potřeby přesnějšího měření by byla vhodnější tolerance 1 % a méně. Důvod, proč vybrat kilo-ohmové hodnoty se nachází ve vzorci pro výpočet výkonu

$$P = U \cdot I$$

Napětí je dáno výstupním rozsahem sondy, tudíž ho nelze ovlivnit, naopak proud je možné snadno změnit za pomoci Ohmova zákona.

$$I = \frac{U}{R}$$

Jak je vidět, závislost proudu na odporu je nepřímo úměrná – s klesajícím odporem roste proud a naopak. Proto je vhodnější použít rezistor s větším odporem, jelikož ním poteče menší proud a bude tak docházet k menším tepelným ztrátám. [28]

2.2.5 NPN zesilovač/spínač

Jak už jsem zmínil v odstavci o tranzistorech 2.2.3, jejich hlavní schopností je zesilování proudu. Existují tři různé způsoby zapojení tranzistoru, zmíním ten nejčastější – zapojení se společným emitorem. Je velmi jednoduché a tranzistor je pouze potřeba doplnit o ochranný rezistor, který ho chrání před proražením. Schéma se nachází na obrázku 2.10. Pin *IN* nízkými proudy spíná spotřebič. Pro výběr vhodného tranzistoru je nutné znát několik údajů, které lze dohledat v datasheetu: [16]

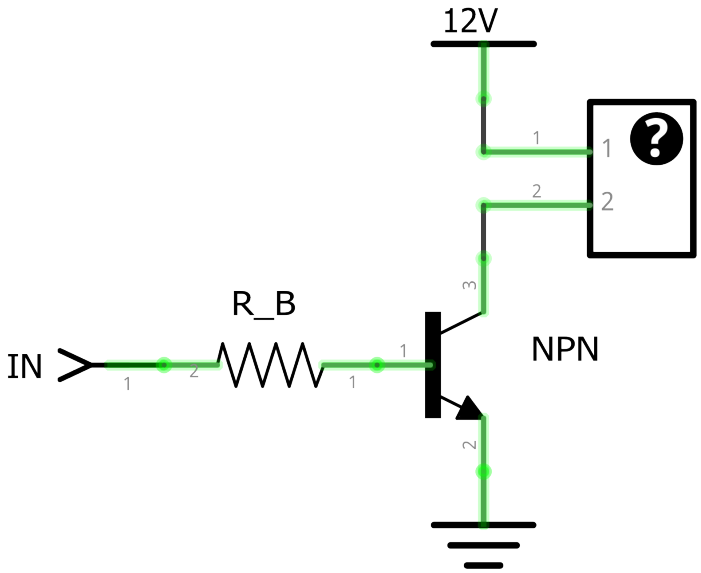
Kolektorový proud I_C hodnotou odpovídá spínanému proudu. Je nutné dodržet horní hranici tranzistoru. [12]

Bázový proud I_B ovlivňuje kolektorový proud. [12]

Proudový zesilovací činitel h_{FE} udává poměr mezi bázovým a kolektorovým proudem. Například hodnota 100 značí, že při změně bázového proudu o 1 mA dojde k nárůstu (či poklesu) kolektorového proudu o 100 mA. [12]

Saturační napětí V_{CE} značí úbytek napětí na přechodu mezi kolektorem a emitorem. Zjednodušeně řečeno je nutné jej odečíst od napájecího napětí. [12]

Úbytek napětí báze – emitor V_{BE} je důležitý při výpočtu R_B. [12]



Obrázek 2.10: NPN zesilovač

Aby nedošlo ke zničení tranzistoru, je nutné omezit bázový proud. K tomu slouží bázový rezistor R_B. Jeho hodnotu určuje Ohmův zákon [12]

$$R_B = \frac{U_{R_B}}{I_B}$$

kde U_{R_B} je úbytek napětí na rezistoru, kterého je potřeba dosáhnout. Ten je závislý na V_{BE} a napětí na pinu IN (V_{IN}). [12]

$$U_{R_B} = V_{IN} - V_{BE}$$

Bázový proud vyhází z h_{FE} a spínaného proudu. Aby bylo jisté, že dojde k sepnutí, používá se trojnásobný proud. [12]

$$I_B = \frac{I_C}{h_{FE}} \cdot 3$$

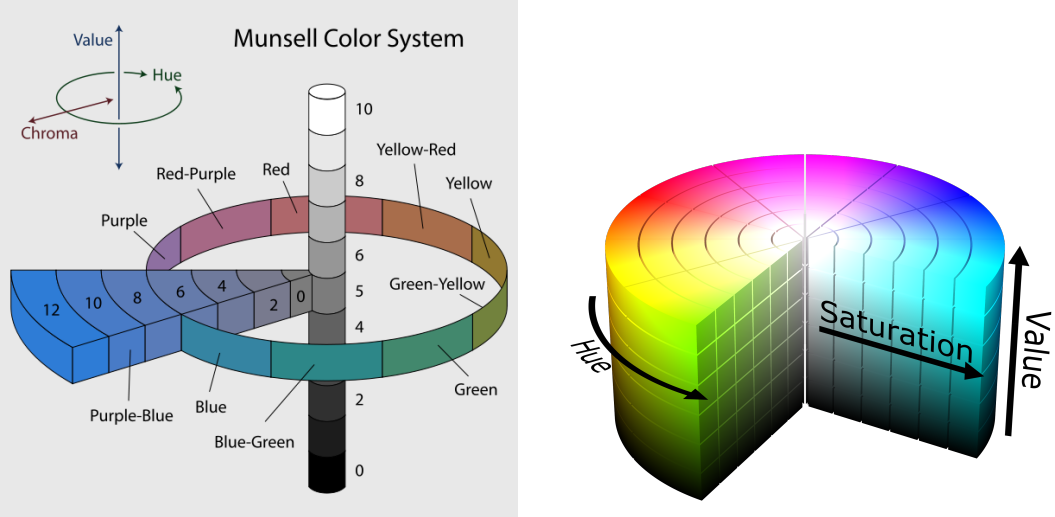
Z výše uvedených rovnic lze sestavit konečnou rovnici pro výpočet bázového rezistoru [12]

$$R_B = \frac{V_{IN} - V_{BE}}{\frac{I_C}{h_{FE}} \cdot 3}$$

2.3 Barevné modely a jejich způsob ukládání

Popsat barvu lze více způsoby. Z hlediska lidského vnímání barev je vhodný Munsellův barevný systém (obr. 2.11(a)), který barvu popisuje pomocí několika vlastností jako je odstín, jas a sytost. Odstín (*Hue*) určuje vlnovou délkou a odlišuje jednu barvu od druhé. Jas (*Value*) definuje jednoduchou stupnici od černé, přes odstíny šedé až po bílou. Sytost (*Chroma*) je barevná škála od šedé po čistý odstín barvy při konstantním jasu. [4]

Další možností jak vytvořit barvu je kombinací několika základních barev, jejichž mícháním vznikají nové barvy. Počet barev, jejich odstín a to, jakým způsobem se míchají určují právě různé barevné modely.



(a) *Jacob Rus, CC-BY-SA 3.0*

(b) *SharkD, CC BY-SA 3.0*

Obrázek 2.11: Munsellův barevný systém a z něj vycházející HSV model

2.3.1 HSV (HSB) a HSL model

Tyto modely vychází z Munsellova popisu barev. Dávají přednost blízkosti k lidskému vnímání barev před digitálním zpracováním, což znamená, že např. monitor s tímto modelem neumí přímo zpracovat a barva tak musí být před vykreslením převedena do jiného modelu. Jako hlavní výhodu lze uvést možnost změnit jednotlivé charakteristiky barvy, aniž by ovlivnily jiné. Výsledná barva nevzniká mícháním základních barev jako u ostatních modelů, nýbrž kombinací odstínu (**H**ue), sytosti (**S**aturation) a hodnoty (**V**alue), což je vlastně jas (občas značen **B**rightness). Model HSL místo jasu využívá světelnost (**L**ightness). [4]

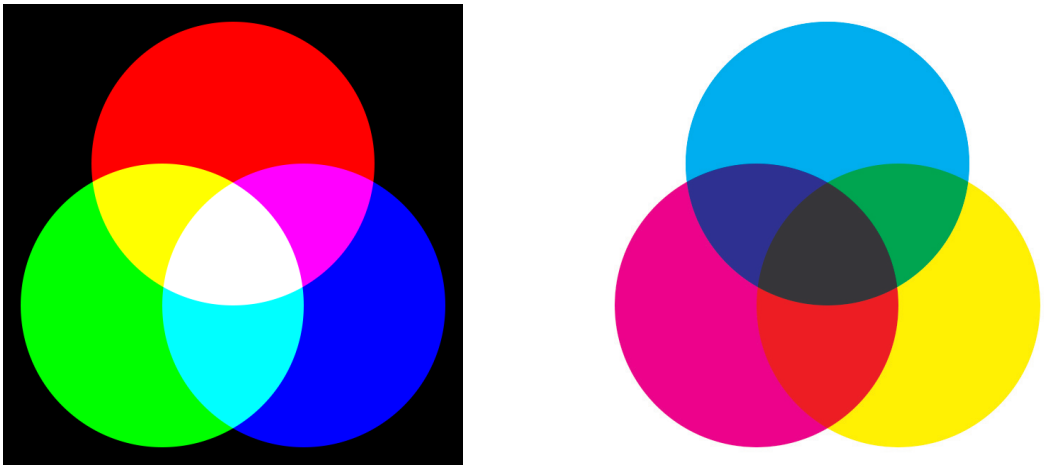
2.3.2 RGB(A) model

Předchozí modely jsou zcela nevhodné pro strojové zpracovaní. Proto bylo potřeba navrhnout jiné, které budou barvy míchat z několika základních barev. Ty jsou u tohoto modelu tři: červená (**R**ed), zelená (**G**reen) a modrá (**B**lue). Někdy může být dále uváděna průhlednost (tzv. **A**lfa kanál). Míchání barev u tohoto modelu je aditivní (jednotlivé složky se sčítají). To znamená, že výsledné smíšené světlo je intenzivnější než jeho jednotlivé složky. Výsledkem smíchání všech tří základních barev vzniká bílá. Tento model tak odpovídá chování běžného denního světla (které je taktéž složeno z několika barev) a proto se využívá u zobrazovacích zařízení (monitory, dataprojektory, televize). [32] Obrázek 2.12 názorně popisuje chování RGB modelu.

2.3.3 CMY(K) model

Model CMY(K) je opakem RGB(A) modelu. Zatímco RGB je aditivní (barvy se sčítají), tak CMY je subtraktivní, tedy barvy se odčítají. To znamená, že smícháním všech barev nevzniká bílá, nýbrž naopak černá. Tento model nalézá využití zejména při tisku.

Model má opět tři základní barvy: azurová (**C**yan), purpurová (**M**agenta) a žlutá (**Y**ellow). V tiskárnách nalezneme ještě černou (klíčová, **K**ey), a to hlavně z důvodu úspory toneru a věrnějšího podání černé. Chování modelu lze vidět na obrázku 2.12. [32]



Obrázek 2.12: RGB (*vlevo*) a CMY (*vpravo*) model

2.3.4 Ukládání barev v informatice

Při ukládání barvy na počítači je nutné se nejdříve určit s jakou přesností je potřeba ji popsat, jakým výkonem disponuje hardware, který s ní bude dále pracovat a co vlastně barva popisuje. Např. pro uchování fotky je vhodnější vyšší počet barev na úkor rychlosti zpracování a pro popis se více hodí RGB model. Naopak v případě tisku lze s ohledem na schopnosti tiskárny omezit celkový počet barev a vybraným modelem bude CMYK.

Přesnost popisu barvy se odvíjí od počtu bitů, které se využívají pro její uchování. Tento počet se nazývá *bitová hloubka*. Čím vyšší je, tím podrobněji je možné barvu popsat, ale na druhou stranu rostou nároky na velikost úložiště a zátěž procesoru při zpracování. [41]

24-bitová barevná hloubka

Dnes nejrozšířenější. Pro uchování jedné barvy potřebuje 3 byty - jeden pro každou složku barvy. Při použití s RGB modelem se barva zapisuje jako šestimístné (2 číslice pro každou barvu) číslo v šestnáctkové soustavě s `#` na začátku. Černou tak lze zapsat jako `#000000`, oranžovou jako `#FF7F00`. [32] Maximální počet barev, které lze touto barevnou hloubkou rozlišit vychází ze skutečnosti, že do jednoho byte lze uložit hodnoty v rozsahu 0 – 255:

$$256 * 256 * 256 = 16\,777\,216$$

Počet barev, které je lidské oko schopno rozeznat nelze přesně určit, ale obecně se udává, že tento počet pro věrnou reprodukci obrazu postačuje, proto je tento model taktéž označován jako *True Color*. Často je také doplněn o dalších 8 bitů pro popis průhlednosti. [41]

8-bitová barevná hloubka

Dnes se nachází spíše na levnějších zobrazovacích zařízeních (vestavěné displeje), u kterých je spotřeba a cena důležitější než vysoká věrnost obrazu. Barva je uložena do jednoho bytu, což znamená, že lze popsat pouze 256 různých barev.

V kombinaci s RGB modelem vzniká potřeba uložit do 8 bitů tři hodnoty – k dispozici jsou tak 3 bity pro dva barevné kanály a jen 2 bity pro třetí kanál. Uspořádání je pak následující: `RRRGGGBB` (lidské oko je nejméně citlivé na modrou barvu, proto tolik nevadí uchování pomocí 2 bitů). [32]

Často se těchto 8 bitů nevyužívá přímo pro popis barvy, ale pouze jako index do palety barev. Ta je buď softwarová, tudíž ji lze měnit podle potřeby aplikace, nebo je uložena přímo v paměti hardwaru a vyjadřuje sadu barev, které je zařízení schopné zpracovat.

Ostatní

Dříve bylo možné narazit na 15-bitovou hloubku, označovanou také jako *Low Color*. Ta pro každou barvu využila 5 bitů, tedy 2 byty celkem (jeden bit zůstal nevyužit). Počet barev pak odpovídal číslu $2^{15} = 32\,768$. Využitím skutečnosti, že lidské oko je nejcitlivější na zelenou barvu, lze nevyužitý bit přiřadit zelené barvě. Vzniklá 16-bitová hloubka se nazývá *High Color* popisující $2^{16} = 65\,536$ různých barev. [41]

V případě, že je i 8-bitová hloubka příliš podrobná (popř. příliš náročná na hardware), tak lze k popisu barvy využít pouze 4 (16 barev) nebo 2 (4 barvy) bity. Ty odkazují na barvy umístěné v tabulce barev. Poslední možností je použít jen jeden bit (zpravidla černá a bílá) též zvaný jako *Mono Color*. [41]

Naopak, při snaze o co nejvěrnější popis, může být i 16 miliónů barev málo. Příkladem takové situace je velmi tmavá scéna. V případě 24-bitové hloubky je k dispozici pouze 256 odstínů šedé, což nemusí být dostatečné. Naopak rozvoj technologií umožňuje zpracovat větší barevné hloubky při vysokém rozlišení, tudíž vznikla např. na 48-bitová barevná hloubka (16 bitů na jeden kanál) nazývaná též *Deep Color*. Samozřejmě existují i další barevné hloubky, ovšem s těmi se dnes téměř nesetkáme. [41]

2.4 Mikrokontrolér

Mikrokontrolér nebo též mikropočítač (anglicky microcontroller či **M**icro**C**ontroller **U**nit, zkráceně MCU) je většinou integrovaný obvod obsahující kompletní počítač. Ovšem nejedná se přímo o počítač v tom smyslu, jak jej chápe většina lidí – osobní počítač (zkráceně PC z anglického *personal computer*) s klávesnicí, myší a obrazovkou, i když mají spoustu společných vlastností. Zatímco u PC je kladen důraz na snadnou komunikaci s uživatelem a vysoký výkon, tak u MCU je to naopak miniaturizace a malý odběr. Informace z okolního světa přijímá v podobě signálů, které jsou tvořeny různými čidly (tlačítka, teploměry, tlakové senzory, apod.). Obdobně probíhá komunikace i druhým směrem, kdy tyto signály mohou být využity k rozsvěcení kontrolek, spínaní motorů, ovládaní displejů, atd.

Využití nachází především jako součást vestavěných (embedded) systémů. Důvodem, proč využít MCU je jednoduchost návrhu systému – není nutné navrhovat složité elektrické obvody, místo toho postačí popsat chování pomocí programovacího jazyku a ten nahrát do MCU. Další z výhod je možnost pozdější úpravy chování, jakožto přidání funkcí nebo oprava chyb.

Aby MCU splnil definici počítače, musí být schopný zpracovávat data pomocí předem vytvořeného programu. K tomu využívá CPU která má k dispozici paměť (popř. několik různých pamětí). Hlavní výhodou MCU jsou dodatečné moduly, které rozšiřují schopnosti CPU o další funkce. [38]

2.4.1 Centrální procesorová jednotka (CPU)

CPU (z anglického *central processing unit*) tvoří jádro MCU, jelikož se stará o vykonávaní programu. Ten je uložen v paměti v podobě posloupnosti jednotlivých instrukcí. Seznam instrukcí, které je CPU schopná vykonat se nazývá *instrukční sada*. [38]

Existují dva hlavní přístupy k vytvoření instrukční sady: [38]

CISC Komplexní instrukční sada (*complex instruction set computing*), se snaží pokrýt velmi široký okruh funkcí, které by jinak bylo nutné realizovat kombinací jednodušších. Díky tomu tato instrukční sada obsahuje až stovky různých instrukcí. To zvyšuje náklady na vývoj a potřebný počet tranzistorů. Každá instrukce trvá rozdílný počet taktů v závislosti na její složitosti. Výhodou CISC je kratší program a tedy menší počet přístupů do paměti.

RISC Redukovaní instrukční sada (*reduced instruction set computing*) naopak cílí na jednoduchou instrukční sadu s minimem instrukcí. Instrukční sada tak zpravidla obsahuje jen několik desítek instrukcí. Vykonání jakékoli instrukce vždy trvá jeden takt. RISC CPU zpravidla obsahují větší počet registrů než CISC.

Struktura CPU je poměrně složitá, mezi hlavní prvky patří: [38]

Aritmeticko-logická jednotka (ALU z *arithmetic logic unit*) provádí logické a matematické výpočty (sčítání, logický součin, ...). Výsledky lze využít k větvení programu.

Registry Jedná se o velmi rychlou paměť, její velikost zpravidla odpovídá šířce slova CPU. Většina CPU neumí pracovat přímo s daty v pamětí, proto je musí nejdříve načíst do registrů, zde je zpracuje (např. pomocí ALU) a modifikovaná data opět uloží do paměti. Mimo to každá CPU obsahuje ještě sadu řídících registrů. Ty slouží k ovládání chování ostatních jednotek CPU.

Řadič Řídí tok data mezi pamětí, registry a ALU, případně i mezi dalšími jednotkami.

2.4.2 Paměť

Dnešní MCU obsahují zpravidla více druhů paměti, které mohou být v závislosti na použité technologii buď trvalé nebo dočasné, nabízejí různé rychlosti čtení a zápisu dat, liší se maximálním počet zápisů a samozřejmě i cenou.

Z historického hlediska existují následující dvě možnosti uspořádání paměti v rámci CPU: [38]

Von Neumannova architektura je velmi jednoduchá. CPU obsahuje pouze jednu paměť společnou pro program i data, tudíž si vystačí jen s jednou sběrnicí.

Harvardská architektura naproti tomu odděluje programovou a datovou paměť. Každá má vlastní sběrnici, díky tomu může být postavena na různé technologii. Další výhodou je možnost paralelizmu. V případě Von Neumannovy architektury nebylo možné v jednu chvíli číst instrukci z programu a zároveň ukládat data. Tyto výhody způsobili, že dnes většina CPU využívá právě Harvardskou architekturu.

Program je uložen v nevolatilní paměti[3] nejčastěji typu FLASH (méně často EEPROM, která je dražší, nebo ROM, která slouží pouze ke čtení a je nutné program nahrát už při výrobě). Proměnné se ukládají v paměti RAM, která je rychlá ale volatilní (po odpojení napájení se smaže). Občas je MCU doplněn ještě třetí nevolatilní pamětí (typu FLASH nebo EEPROM) pro uchování uživatelského nastavení. V případě, že tomu tak není a je třeba uchovat určitá data i po vypnutí, je nutné MCU doplnit o externí paměť. [38]

[3] Nevolatilní paměť potřebuje napájení pro sboji činnost (čtení, zápis), ale při odpojení napájení se informace uchová.

2.4.3 Další moduly

Klíčovou vlastností MCU jsou rozšiřující moduly. Existuje jich velké množství a záleží tak na výrobci, které a v jakém počtu se rozhodne integrovat. Obsažené moduly proto slouží jako hlavní kritérium při výběru MCU v závislosti na plánovaném použití. Zde je seznam několika nejčastějších: [38]

Vstupně/výstupní porty slouží ke komunikaci CPU s okolím. Umožňují číst signály ze vstupních pinů nebo naopak vysílat signály na výstupních pinech. Piny se sdružují do portů, které jsou v rámci CPU realizovány jako registry. Každý port je většinou zastoupen dvěma registry, kde jeden registr slouží k řízení směru (každý jeho bit určuje zda je konkrétní pin vstupní nebo výstupní) který lze zpravidla měnit i za běhu a druhý registr slouží ke čtení/zápisu hodnoty na/z pinu. Registrů může být i více, je-li třeba ovlivnit další vlastnosti pinu (příkladem může být připojení pull up rezistoru, viz dále). [36]

Signály jsou kódovány binárně pomocí napětí, proto je nutné definovat dvě logické úrovně:

- **Vysokou** pro logickou jedničku, ta je velmi blízká napájecímu napětí.
- **Nízkou** pro logickou nulu, představuje 0 V, tzv. referenční zem.

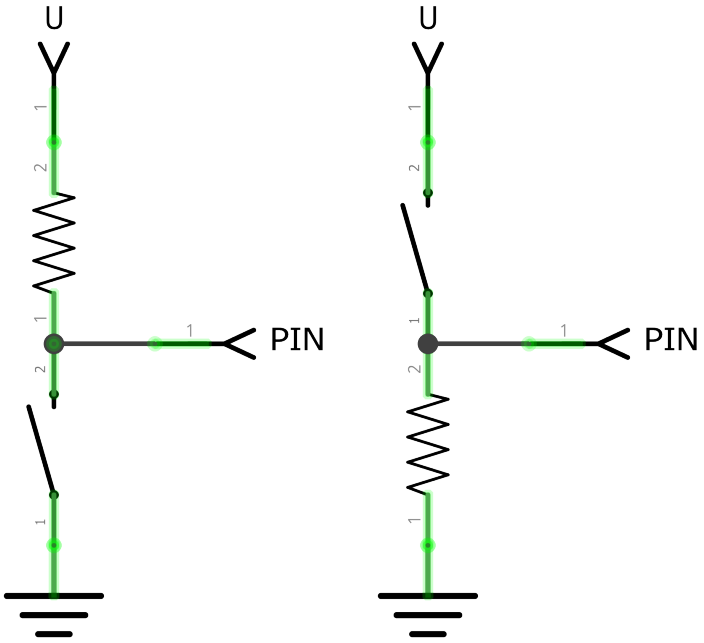
Tyto hodnoty obsahují určitou toleranci (cca 30 %), proto jsou problémové napětí rovné přibližně polovině hodnoty vysoké úrovně. Ty leží v tzv. zakázaném pásmu a není definováno, zda představují nulu nebo jedničku. Obdobný problém tvoří nezapojené vstupní piny (typicky připojené tlačítko). Ve chvíli, kdy je čtena hodnota z pinu aniž by byl někam zapojen, tak slouží jako anténa a vzniká na něm šum, proto není možné říct, jakou hodnotu MCU přečte a zároveň se tato hodnota s časem mění. [36]

Aby se tomuto nechtěnému jevu zabránilo, je nutné na pin připojit tzv. *pull up* nebo *pull down* rezistor. Tento rezistor je připojen na vstupní napětí (v případě pull down rezistoru na zem) a vytváří trvale logickou jedničku (u pull down nulu). Hodnota rezistoru se volí tak, aby protékající proud byl s ohledem na spotřebu co nejmenší (typicky několik kilo-ohmů). Spousta MCU je vnitřně vybavena sadou pull up rezistorů, které lze programově připojit na piny (u levnějších MCU to je možné jen na některých pinech). [36]

V určitých situacích může být potřeba výstupní pin odpojit a zajistit tak, aby nenabýval ani jednoho ze svých dvou stavů realizovaných jako připojení k zemi nebo k napájecímu napětí. Toho lze docílit pouze tím, že se změní na vstupní. Tím se dostane do stavu tzv. *vysoké impedance*, což značí, že pin je odpojený. [36]

A/D převodník nebo-li analogově digitální převodník slouží k převodu spojitého (analogového) signálu na diskrétní (digitální). Pro svoji činnost potřebuje tzv. *referenční napětí*, vůči kterému probíhá měření. Výsledkem převodu tudíž není konkrétní číslo, nýbrž ukazatel do rozsahu 0 – U_{REF}. MCU umožňuje výběr z více různých referenčních napětí, případně je vybaven pinem pro přivedení vlastního externího zdroje referenčního napětí. [34]

Převodník pracuje s určitým *rozlišením*, tedy je schopný měřit s nějakou přesností. Ta se vyjadřuje v počtu bitů, který je potřeba pro uložení výsledku. Běžné jsou 8 a 10-bitové převodníky, v případě vyšší přesnosti lze ale sehnat i 24-bitový. [34]



(a) Pull down (b) Pull up

Obrázek 2.13: Zapojení pull up a pull down rezistoru

Z rozlišení a referenčního napětí lze snadno určit přesnost měření. Nejdříve je nutné vypočíst počet různých vzorků, které lze převodníkem změřit. K tomu slouží následující vztah

$$2^n$$

kde n odpovídá rozlišení převodníku. Nejmenší měřitelný rozdíl pak je

$$U_{REF} \div 2^n$$

Ze vztahu je patrné, že i nepatrné zvýšení rozlišení výrazně (kvadraticky) zvyšuje přesnost. To stejné lze říct i o snížení referenčního napětí, i když zde se jedná pouze o lineární závislost. [34]

Jak je zmíněno na začátku, A/D převodník nezíská konkrétní hodnotu napětí (ani nemůže), pouze určí poměr mezi referenčním a měřeným napětím. Např. výše uvedený 10-bitový převodník s referenčním napětím 5 V převede vstupní napětí 2,75 V na číslo 563. Tuto hodnotu je nutné přemapovat pomocí vzorce [34]

$$U_{MERENE} = \frac{U_{REF}}{rozliseni} \cdot namerena\ hodnota$$

Po dosazení čísla 563 z příkladu vyjde napětí

$$U_{MERENE} = \frac{5}{1024} \cdot 563 = 2,749\,\mathrm{V}$$

D/A převodník funguje na velmi podobném principu jako A/D převodník, pouze probíhá převod opačným směrem – z digitálního signálu tvoří analogový. Platí zde stejné vlastnosti jako rozlišení převodu, potřeba přemapovat hodnoty, apod. [34]

Řadič přerušení CPU zpracovává program sekvenčně, tedy postupně jednu instrukci za druhou, což se v určitých situacích může jevit jako poměrně nevhodné chování. Příkladem může být obsluha zmáčknutí tlačítka. Z pohledu uživatele je třeba, aby se akce vykonala okamžitě. Což znamená, že by programátor musel přibližně každých 50 – 100 ms kontrolovat, zda není tlačítko zmáčknuto. Aby nebylo nutné to dělat takto složitě, je CPU doplněna o tzv. *přerušení*. Zjednodušeně řečeno, nastane-li přerušení, tak CPU přeruší vykonávaní programu, obslouží přerušení a po té opět pokračuje v programu na místě, kde skončila. [36]

Přerušení jsou programovatelná a mohou je vyvolat různé moduly, ale i např. ALU. V případě modulů se může jednat o změnu úrovně na vstupním pinu, přijatá data na některém z komunikačních modulů, dokončený převod na A/D převodníku, časovač a další. Obsluhu přerušení musí povolit a naprogramovat programátor – ve výchozím nastavení jsou většinou vypnutá. Obsluha přerušení je klasická funkce, která by měla být co nejjednodušší aby její vykonání nezabralo více jak několik ms. [38]

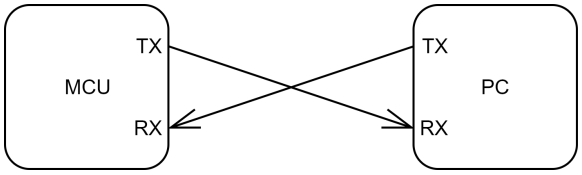
Časovač Časovač je velmi důležitý modul, který umožňuje CPU přenechat poměrně náročné počítání času externímu modulu a věnovat se samotnému programu. MCU je zpravidla vybaven více časovači s různou přesností a rozsahem. Ty jsou schopny vykonávat více různých funkcí. Nalezneme v nich např. čítače, které počítají počet ms (popř. μs) od spuštění, časovače generující přerušení (vhodné, chceme-li na chvíli uspat MCU) či generátory PWM pulzů. [37]

Různé časovače využívají různou technologii (čím přesnější má časovač být, tím vyšší náklady jsou spojené s jeho výrobou a roste spotřeba energie [35]) pro generování impulzů a stejně tak běží na různých (obvykle nastavitelných) frekvencích. Záleží proto především na programátorovi, k jakému účelu chce časovač využít a podle toho je třeba vybrat vhodnou technologii a frekvenci. [37]

Watchdog Watchdog v překladu z angličtiny znamená *hlídací pes*, což poměrně přesně popisuje funkci tohoto modulu. Jeho jediným, ale důležitým úkolem je hlídat, zda program běží, nebo se dostal do nekonečné smyčky. Funguje na principu periodického (zpravidla na začátku hlavní smyčky) nulování čítače tohoto modulu. Ten je zároveň napojen na zdroj hodin, které ho periodicky zvyšují. V případě, že dojde k přetečení čítače (program ho po určitou dobu nevynuloval), tak watchdog restartuje systém. Hlavní smysl existence watchdogu je ten, aby byl systém schopný se zotavit z chyby a pokračovat v činnosti bez potřeby zásahu uživatele. [29]

UART UART je označení pro komunikační rozhraní mezi dvěma zařízeními někdy též nazýváno jako sériová linka či sériový port. Jedná se o zkratku slov *Universal Asynchronous Receiver and Transmitter*, v překladu *univerzální asynchronní přijímač a vysílač*. Pro komunikaci využívá dva vodiče: `TX` z anglického *transmit* (vysílat) pro odchozí data a `RX` podle *receive* (přijímat) pro příchozí data. Tyto vodiče se zapojují do kříže, jak je znázorněno na obrázku 2.14. [13]

UART využívá asynchronní komunikaci, tudíž jednotlivá zařízení nemají synchronizovaný hodinový signál, ale každé si generuje vlastí. Na jednu stranu to přináší možnost komunikace po jednom vodiči (není nutný speciální vodič pro hodinový signál), ale na druhou stranu je nutné provádět synchronizaci hodinového signálu na obou zařízeních, který navíc musí běžet na stejné frekvenci. [38]

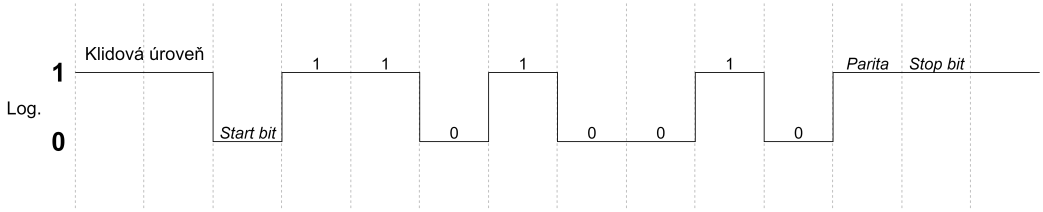


Obrázek 2.14: Spojeni dvou zařízení pomocí UART

Komunikace probíhá následovně. Na začátku je signál v tzv. *klidové úrovni*, která je reprezentována jako logická 1. Přechodem do log. 0 dojde k synchronizaci hodin obou zařízení a zároveň se jedná o signál pro zahájení komunikace nazývaný *start bit*. Následně jsou postupně odvysílány datové bity. Těch může být podle nastavení 5 – 10. Nejběžnější je přenos po 8 bitech. Za datovými bity může následovat *parita* (kontrolní bit pro odhalení chyby). Na závěr je odeslán jeden nebo dva *stop bity*, které slouží k obnovení klidové úrovně, tedy logické 1 a zároveň umožňují přijímači zpracovat příchozí data a připravit se na další komunikaci. [13]

Přenosová rychlost se udává v *baudech* (zkratka Bd) vyjadřujících počet přenesených bitů za sekundu a odpovídá frekvenci hodinového signálu. Efektivní přenosová rychlost je o něco menší z důvodu přítomnosti režijních bitů (start bit, parita, ...). [13]

Všechny výše uvedené vlastnosti jako přenosová rychlost, počet přenášených bitů, parita či stop bity je třeba nastavit na obou zařízeních ještě před zahájením komunikace. Na obrázku 2.15 je znázorněn přenos 8 bitů (`11010010`) pomocí UART s lichou paritou a jedním stop bitem.



Obrázek 2.15: Přenos dat pomocí UART

2.4.4 Pracovní režimy

Aby odběr MCU byl co nejnižší, obsahuje zpravidla několik režimů, v kterých se může nacházet. V základním režimu je aktivní CPU a všechny moduly, odběr se zpravidla pohybuje v řádu jednotek až desítek mA. Protože často MCU nic nedělá a pouze čeká na interakci od uživatele, byl doplněn o několik různých režimů spánku. V těchto režimech je deaktivované CPU a jsou aktivní pouze moduly, popř. jen některé z modulů. V závislosti na tom, o jak hluboký spánek se jedná, klesá spotřeba až na desetiny μA. K probuzení slouží přerušení, přičemž rychlost probuzení se odvíjí od hloubky spánku ve které se MCU nacházel, ale stále se jedná maximálně o ms. [35]

2.4.5 Programování a vývoj

Dnes se programy pro MCU píší velmi podobně jako na PC. Většinou se používá jazyk C, popř. C++, ale prakticky lze program napsat v libovolném jazyce. Pouze musí existovat

překladač pro daný jazyk a mikrokontrolér. Dnešní MCU používají přístup ISP (*In-System Programming*), tedy umožňují nahrání programu i do osazeného zařízení. Komunikace probíhá většinou pomocí UART. Aby bylo možné využít ISP, tak MCU musí obsahovat *bootloader*, což je program, který v případě potřeby přijme data, uloží je do trvalé paměti a spustí je jako program. Mimo to může provádět i další činnost, např. inicializaci modulů a paměti. Některé MCU obsahují také ladící rozhraní pomocí kterého je možné za běhu ladit program obdobně jako při krokování programu na PC. [39]

Jinou možností testování je simulátor – program na PC schopný simulovat chování MCU. Není tak nutné fyzicky vlastnit zapojený MCU včetně periferií a nehrozí jeho (jejich) poškození. Takový program je ale velmi složité a nákladné vytvořit. Dnešní MCU jsou velmi komplikované a simulace případných neobvyklých komponent použitích v zapojení může být nemožná. Proto simulátory existují jen pro několik modelů a dnes se od nich odpouští. [39]

Naopak na vzestupu jsou *vývojové desky*. Tedy naletovaný mikrokontrolér v základním propojení s několika základními periferiemi (tlačítka, diody, napájení, USB převodník, apod.) zpravidla doplněný o vývody pro snadné zapojení do nepájivého pole. Tyto kity jsou vhodné pro seznámení s konkrétním MCU a ladění zařízení před jejich finálním návrhem. Dnes je zřejmě nejznámější projekt Arduino, který shlukuje několik takových vývojových desek obsahujících MCU ATmega od firmy AVR. Ty se liší velikostí, přídavnými periferiemi, modelem MCU i cenou. Jinou variantou jsou vývojové desky používající rodinu mikrokontrolérů STM32 od firmy STMicroelectronics. [39]

2.5 Principy řízení světelných zdrojů a PWM

Při řízení světelných zdrojů lze využít v podstatě dva hlavní přístupy. Oba mají své výhody i nevýhody a záleží tak především na typu svítidla, které má být ovládáno.

2.5.1 Využití Ohmova zákona

Velmi jednoduchý způsob, jak ovlivnit výkon svítidla. Vychází ze vzorce pro výpočet výkonu $P = U * I$ a Ohmova zákona $U = R * I$. Kombinace definuje závislost výkonu na proudu, napětí a odporu:

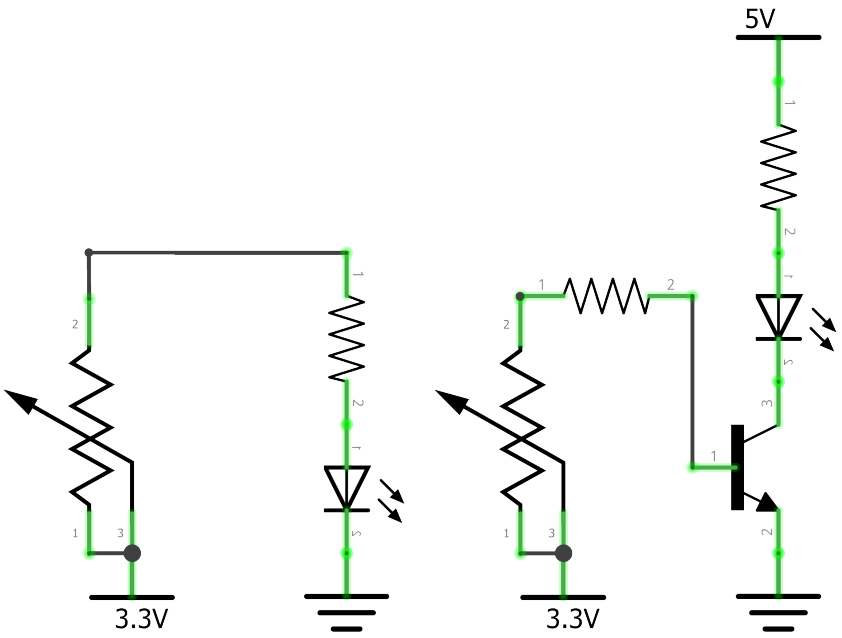
$$P = I^2 * R = \frac{U^2}{R}$$

Napětí dodávané zdrojem je konstantní, protékající proud je daný použitým světelným zdrojem, takže jediné, co lze změnit je odpor.

K tomu slouží tzv. proměnný rezistor, který je realizován pomocí tří různých součástek. Všechny tři pracují na stejném principu – posun jezdce po odporové dráze. Nejstarší a dnes téměř nepoužívaný je *reostat*. Ten využívá namotaný odporový drát na válci po kterém se posouvá jezdec. Reostat nahradil *potenciometr*. Především proto, že je menší. Konstrukce je obdobná, odpor se mění pohybem jezdce po odporové dráze (většinou uhlíkové). Z důvodu potřeby další miniaturizace existují ještě tzv. *trimry*. Jedná se v podstatě o miniaturizované potenciometry bez hřídele. Hodnota se nastavuje za pomoci šroubováku a to zpravidla jen jednou při kompletaci zařízení. [15]

Tyto proměnné rezistory mají omezený maximální výkon, proto je nutné doplnit zapojení o jednoduchý NPN zesilovač.

Výhodou tohoto zapojení je jednoduchost, ale převládají nevýhody. Pokud by k řízení měl být využit MCU, musel by být vybaven D/A převodníkem, což u těch levnějších není standart. Jako další z nevýhod lze zmínit nemožnost takto řídit veškeré světelné zdroje.



(a) Základní zapojení (b) Zapojení s NPN tranzistorem

Obrázek 2.16: Nejjednodušší možná zapojení pro ovládání osvětlení

Omezím-li se na ty běžně používané v domácnosti, tak je tímto způsobem možné regulovat žárovky (a všechny zdroje ve kterých se žhaví vlákno). Naopak ho nelze využít pro ovládání zářivek (včetně kompaktních a podobných svítidel s trubicí) a LED (takže by výše uvedené zapojení ani nefungovalo). U zářivek je pro stmívání nutné použít speciální elektronický předřadník (tímto tématem se dále už zabývat nebudu). V případě LED narazíme problém s tzv. *prahovým napětím*, tedy hranici tvořenou PN přechodem (obdobně jako u klasické usměrňovací diody 2.2.2), kterou je nutné překonat, aby diodou začal protékat proud. Po jeho překročení je nárost proudu (a tedy i jasu) poměrné rychlý a brzy dosáhne maxima, tudíž plynulá regulace je tímto způsobem téměř nemožná.

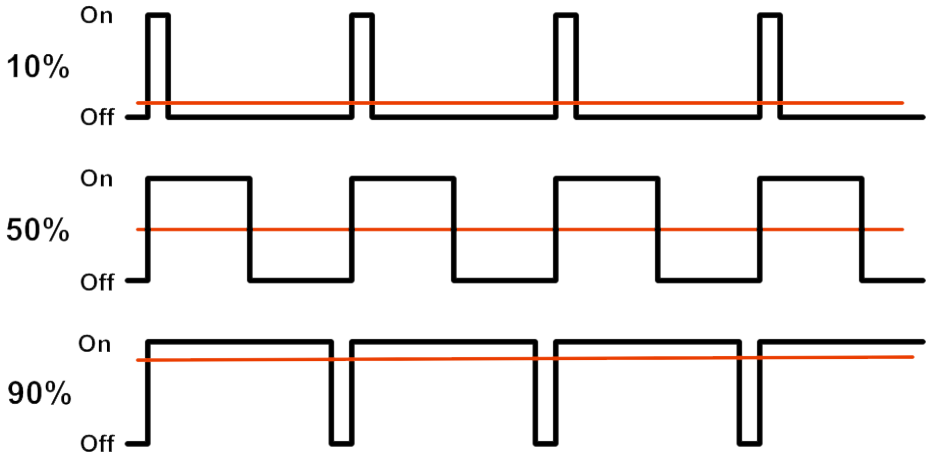
2.5.2 Pulzně šířková modulace (PWM)

V tomto případě není proud nijak omezován (např. pomocí rezistoru), nýbrž je distribuován v podobě pulzů do spotřebiče. Většina zdrojů může nabývat jen dvou stavů – zapnuto a vypnuto. Této skutečnosti právě využívá pulzně šířková modulace. Postačí, jsou-li tyto stavy měněny dostatečně rychle a díky setrvačnosti splynou do jednoho. A je jedno, zda se jedná o mechanickou setrvačnost motoru nebo setrvačnost lidského oka. Proto, bude-li ve výstupních signálu převládat zapnutý stav, vznikne dojem, že světlo svítí a v opačném případě (tedy, kdy převládají stavy vypnuto) naopak bude světlo vypadat zhasnutě, popř. bude svítí slabě. [37]

Poměr těchto stavů se nazývá *střída*. Délka střídy (trvání stavu zapnuto a vypnuto) se nazývá *perioda*. V případě osvětlení musí být perioda vyšší než frekvence, kterou pracuje lidské oko. Jen tak dojde k splynutí jednotlivých stavů a díky nedokonalosti oka vznikne dojem, že svítidlo mění svůj jas. [37]

Výhodou tohoto řešení je snadná realizace pomocí MCU, spousta jich přímo obsahuje časovače schopné generovat PWM signál, což znamená jednodušší práci pro programátora a

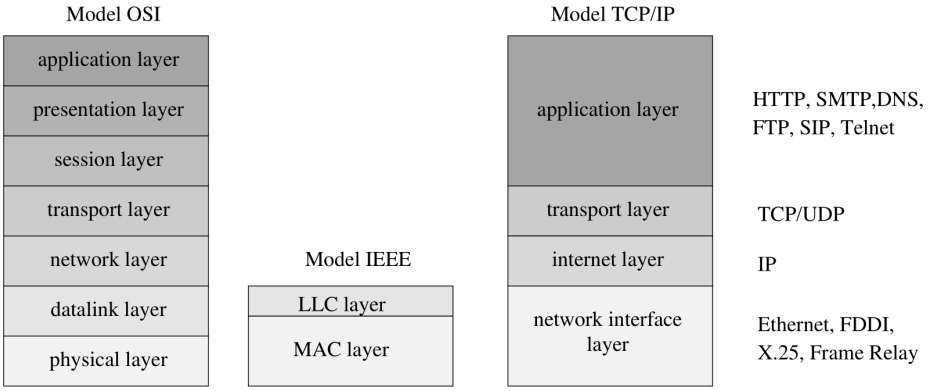
minimální nároky na procesor – o vše se stará samotný modul. [37] Mimo to je tato regulace zcela plynulá a lze ji použít jak pro žárovky (zde může dojít k problémům se setrvačností rozehřátého vlákna, které nemusí dostatečně rychle zchladnout) tak pro LED, jelikož po celou dobu je napětí vyšší než prahové.



Obrázek 2.17: PWM signál s různou střídou [33]

2.6 Počítačové sítě a související technologie

Pro popis počítačové sítě se používá referenční model je ISO/OSI, který dělí síť do sedmi vrstev. Každá z nich využívá služby o řád nižší vrstvy a naopak sama poskytuje služby o řád vyšší vrstvě. Tento model nebyl nikdy zcela implementován, ale posloužil jako vzor pro další protokoly a modely, např. TCP/IP. [30]



Obrázek 2.18: Porovnání modelů ISO/OSI, IEEE a TCP/IP [30]

1. **Fyzická vrstva** nejnižší vrstva, která specifikuje fyzickou komunikaci. Zajišťuje převod proudu bitů na nejčastěji elektrické signály. Tato vrstva dále stanovuje vlastnosti rozhraní, jako napěťové úrovně, kmitočty, modulace ale i např. tvary a a barvy konektorů nebo maximální délku kabelů. [30]

2. **Linková vrstva** zajišťuje komunikaci mezi dvěma nebo více uzly. Definuje kódování dat z vyšších vrstev do rámců, které jsou adresovány pomocí MAC adresy, což je jedinečná 48-bitová adresa konkrétního síťového zařízení. [30]
3. **Síťová vrstva** se stará o směrování v rámci sítě. Nejznámější protokol této vrstvy, je Internet Protocol, tedy IP. Existuje ve více verzích, přičemž dnes je zatím stále nejrozšířenější IPv4, ale pomalu se již přechází na novější IPv6. [30]

   Internet Protocol posílá data po blocích, tzv. *datagramech*. Ty jsou směrovány ze zdrojového počítače skrze jednu či více IP sítí do cílového počítače adresovaného pomocí IP adresy. Ta je v případě IPv4 32-bitová, IPv6 používá 128-bitové adresy. Každý paket je doplněn o metadata obsahující adresu zdroje a cíle, kontrolní součet pro detekci chyb, informaci o pořadí, aj. [30]

   Internet Protokol nezaručuje správné doručení paketů a proto je označován za nespolehlivý. Potřebuje-li aplikace spolehlivý přenos, je nutné ho implementovat pomocí některé z vyšších vrstev. [30]
4. **Transportní vrstva** slouží k přenosu dat přímo mezi aplikacemi na jednotlivých počítačích. Zatímco počítače jsou adresovány IP adresou, tak jednotlivé aplikace jsou identifikovány pomocí čísla portu, který může v jednu chvíli využívat vždy maximálně jedna aplikace. [30]

   Aby byla zajištěna správná funkčnost napříč různými aplikacemi a systémy, byl vytvořen seznam portů a služeb, které ho využívají. Ten dělí porty do tří skupin. Rozsah 0 – 1023 je vyhrazen pro nejběžnější služby. Např. na portu č. 80 běží HTTP. Porty 1024 – 49151 by se měli registrovat u organizace ICANN[4]. Zbylé porty, tedy 49152 – 65535 lze využívat libovolně. [30]

   TCP nebo-li *Transmission Control Protocol* je dnes nejpoužívanějším protokolem transportní vrstvy používaným v Internetu. Protokol zajišťuje spojení mezi dvěma aplikacemi, přičemž garantuje, že obousměrně přenášená data budou spolehlivě doručena ve správném pořadí. Tento protokol využívá např. HTTP nebo e-mail. [30]

   Aby byla zajištěna spolehlivost, je nejdříve nutné navázat spojení mezi aplikacemi. Až poté může být zahájen přenos dat, přičemž jsou data rozděleny do segmentů, každý segment je doplněn TCP hlavičkou a dále předán IP k odeslání. Přijaté segmenty jsou zkontrolovány na správnost pomocí kontrolních součtů, seřazeny do správného pořadí a je potvrzen jejich přijetí. V případě, že by u některého z těchto segmentů nedošlo k potvrzení doručení, je poslán znovu. [30]

   UDP (*User Datagram Protocol*) je oproti TCP označen jako nespolehlivý. A to z toho důvodu, že nijak neručí za to, zda budou přenášená data doručena k cíli ani zda dorazí ve správném pořadí. Před odesláním dat není ani nutné navázat spojení. Odesílaná data jsou posílány v tzv. datagramech, přičemž každý tento datagram je doplněn o jednoduchou hlavičku se zdrojovým a cílovým portem, délkou dat a kontrolním součtem. [30]

[4] Internet Corporation for Assigned Names and Numbers

Hlavní výhodou UDP je nižší režie při přenosu dat, čehož se využívá v aplikacích, u nichž může dojít k menšímu výpadku dat, ale je klíčová nízká odezva (mediální vysílání, on-line hry). [30]

5. **Relační vrstva** udržuje spojení mezi komunikujícími aplikacemi. To znamená, že se stará o vytvoření a zrušení relace, synchronizaci, apod. [30]
6. **Prezentační vrstva** má za úkol převod přijatých dat do tvaru vhodného pro danou aplikaci. Díky tomu se aplikace již nemusí start o správný formát dat, ale pouze o jejich význam. Příkladem převodu může být například kódování řetězců (ukončení, speciální znaky, apod.), převod složitých struktur na řetězce a zpět, komprese a dekomprese dat či např. šifrování. [30]
7. **Aplikační vrstva** Nejvyšší a poslední vrstva modelu ISO/OSI. Na této vrstvě již běží samotné procesy a aplikace. [30]

Wi-Fi

Pojem Wi-Fi je označení pro sadu standardů IEEE 802.11, které popisují bezdrátový přenos dat v počítačových sítích. V ISO/OSI modelu je Wi-Fi umístěno na druhé linkové vrstvě.

K přenosu dat se využívá tzv. bezlicenční pásmo[5] o frekvenci 2.4 GHz. Standart IEEE 802.11 byl postupně rozšiřován. Tato rozšíření se značí jako písmenko na označením standardu. Mezi nejrozšířenější tak patří 802.11b, 802.11g a 802.11n, často zkráceně označované také jako 802.11b/g/n přičemž je zachována zpětná kompatibilita. Později přibyl standart 802.11a, který využívá 5 Ghz pásmo. To je méně zarušené a poskytuje vyšší rychlosti. Tento standard byl dále rozšířen na 802.11ac a 60GHz 802.11ad. [24]

SSID

Každá Wi-Fi síť musí mít identifikátor, tedy SSID (*Service Set Identifier*), což je řetězec složený z ASCII znaků o maximální délce 32 znaků. Toto SSID je broadcastem šířeno do okolí a každý si ho může zobrazit (toto chování lze většinou v nastavení vypnout). [24]

Zabezpečení

Jednoduché zabezpečení sítě se dá vytvořit pomocí skrytí SSID (tedy vypnutí jejího šíření do okolí) a filtrováním povolených MAC adres. Oba tyto kroky může potenciální útočník snadno překonat a proto se nejedná o dostatečné zabezpečení a je nutné ho doplnit (nebo nahradit) autentizací. Postupným vývojem vzniklo několik protokolů: [24]

WEP je nejstarší, dnes již prolomené zabezpečení a jeho má smysl pouze v kombinaci s velmi starým zařízením, které nepodporuje novější protokoly.

WPA vznikl jako rychlá náhrada za prolomené WEP, ale v dnešní době již také není vhodné jej používat.

WPA2 je dnes nejnovější protokol, zatím považovaný za bezpečný.

[5] Bezlicenční pásmo je označení pro určité frekvence, v kterých lze vysílat bez nutnosti hradit poplatky.

2.7 HTTP protokol a související technologie

Již z názvu (Hypertext Transfer Protocol) je patrná hlavní funkce tohoto protokolu. Tím je přenos hypertextových dokumentů (nejčastěji ve formě HTML) v rámci internetu. Obvykle používá TCP port 80, ovšem je možné použít i jiný port popř. i jiný internetový protokol, jako je například UDP.

2.7.1 Princip komunikace

Protokol funguje na principu dotaz – odpověď. Komunikaci zahajuje klient (nejčastěji uživatel pomocí webového prohlížeče) navázáním spojení na portu 80. Následně odešle serveru dotaz kódovaný jako ASCII text. Server na tento dotaz odpoví taktéž formou čistého textu. V závislosti na verzi protokolu se o ukončení spojení stará server nebo klient.

Z popisu komunikace vyvstává nevýhoda HTTP protokolu. Na každý dotaz je otevřeno nové TCP spojení. To jednak zvyšuje režii spojenou se samotným TCP spojením, což je problém zejména v případě rozsáhlých webových stránek složených z většího množství obrázků, skriptů, stylů apod. Tento problém řeší nejnovější verze HTTP/1.1 (ovšem bylo ho možné řešit i ve starší verzi HTTP/1.0). Dalším problémem je to, že server nedokáže určit, zda nový dotaz souvisí s tím předchozím. Protokoly s touto vlastností se označují *bezstavové*, to znamená, že nejsou schopny uchovávat stav komunikace a jednotlivé dotazy, byť od stejného uživatele, tak spolu nijak nesouvisí. Tato vlastnost je poměrně velkou překážkou pro větší weby, např. z důvodu potřeby uchovat uživatele přihlášeného. Proto byl protokol rozšířen o tzv. cookies (viz níže v kapitole 2.7.3). [31]

2.7.2 Verze HTTP

První veřejná verze HTTP 0.9 byla publikována v roce 1991. V současnosti je nejrozšířenější verzí HTTP/1.1 z roku 1999, která je postupně nahrazována rychlejší a modernější verzí HTTP/2.0. Přičemž novější verze je vždy zpětně kompatibilní s tou předchozí. [31]

- **HTTP 0.9**
  Jedná se o velmi jednoduchý, dnes již nepoužívaný protokol, který ještě nepoužívá HTTP hlavičky.

  Komunikaci zahajuje klient odesláním žádosti GET. Ta je složena z klíčového slova GET, mezery, adresy dokumentu (bez adresy serveru či čísla portu) zakončená pomocí CR LF. Příklad: `GET /index.html`. Odpovědí na tento dotaz je žádaný soubor v jazyce HTML ve formě streamu ASCII znaků. Po úspěšném přenosu server uzavře spojení.[9]

  Dojde-li k chybě, je zaslána stejně jako soubor ve formě textu. Neexistuje tak žádný způsob, jak zjistit, zda byl dotaz úspěšný. Tuto informaci lze pouze vyčíst z obsahu odpovědi.[9]

- **HTTP/1.0**
  Největším přínosem této verze jsou HTTP hlavičky. K metodě GET přibyly metody HEAD a POST. [31] Dále je možné přenášet i jiné soubory, než jen čisté ASCII, pomocí standartu MIME[6]. [11]

[6] Jedná se internetový standart pro přenos jiných dat než čistý ASCII text původně navržený pro elektronickou poštu. Pomocí tohoto standartu lze odesílat texty v různém kódování (např. UTF-8), které může být použito i v hlavičce, či přímo binární data. [30]

Komunikace je podobná verzi 0.9. Rozdíl je ve tvaru žádosti. Ta na prvním řádku obsahuje identifikaci metody, adresu dokumentu a nově i verzi HTTP. Na dalších řádcích jsou vloženy jednotlivé hlavičky (verze prohlížeče, podporované formáty, jazyk, cookies, atd.) ve tvaru `Jméno: hodnota` zakončené `CR LF`. Tyto hlavičky jsou od těla zprávy odděleny prázdným řádkem. Metode GET a HEAD mají tělo prázdné, metoda POST v těle uchovává odesílaná data. [11]

Příklad dotazovací hlavičky:

```
GET /index.html HTTP/1.0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept-Language: cs-CZ,cs;q=0.9
```

Obrázek 2.19: Ukázka HTTP/1.0 dotazovací hlavičky

Metody HEAD a POST pracují obdobně. Metoda HEAD funguje na stejném principu jako GET s tím rozdílem, že odpovědí je pouze hlavička bez těla. Toho lze využít například pro zjištění, zda došlo ke změně souboru bez toho, aniž by byl samotný soubor přenášen. V případě metody POST jsou k odesílané hlavičce přibaleny data, které má server zpracovat. Většinou se jedná o hodnoty z formuláře, ale může to být i cokoli jiného (obrázek, video, ...). [31]

Odpověď je také rozšířena o HTTP hlavičku. První řádek obsahuje verzi HTTP a za ním stavový kód odpovědi spolu s jeho názvem. Následují opět jednotlivé hlavičky, od těla oddělené prázdným řádkem.

Příklad odpovědi:

```
HTTP/1.0 200 OK
Date: Mon, 05 Mar 2018 15:48:54 GMT
Server: Apache/1.3.29 (Unix) PHP/4.3.8

<!DOCTYPE html>
...
```

Obrázek 2.20: Ukázka HTTP/1.0 odpovědi

Stavové kódy slouží k identifikaci úspěšnosti dotazu či typu chyby. Je jich několik desítek členěných do 5 kategorií podle typu informace, kterou nesou. Jedná se o třímístné číslo, kdy první číslice identifikuje kategorii a další dvě číslice konkretizují situaci. [11]

1xx Informační kódy Informují o stavu serveru, např. kód `100 Continue` informuje, že server obdržel hlavičku požadavku a čeká až klient odešle tělo zprávy.

2xx Úspěch Požadavek by úspěšně vyřízen. Nejčastější odpovědí je kód `200 OK` následovaný daty.

3xx Přesměrování Tyto kódy slouží k přesměrování požadavků či dokumentů. Lze využít např. při přesunu serveru. Zajímavý je kód `304 Not Modified`, který říká, že soubor nebyl od posledního dotazu změněn a není ho nutné znovu zasílat, čímž se snižuje datový tok.

4xx Klientské chyby Klient odeslal chybnou žádost. Příkladem může být špatně zapsaná adresa – kód `404 Not Found`.

5xx Serverové chyby K chybě došlo až po úspěšném zpracování dotazu a indikuje chybu na serveru. Např. z důvodu přetížení – `503 Service Unavailable`.

- **HTML/1.1**
  Dnes nejrozšířenější verze změnila možnost udržet TCP spojení (udržované spojení) z volitelné na implicitní. [31] To znamená, že zatímco ve verzi 1.0 bylo nutné tuto skutečnost uvést do hlavičky, tak ve verzi 1.1 není spojení ukončeno po přenosu prvního souboru, ale až po uplynutí zadané doby nebo jeho ukončení ze strany klienta. Díky tomu není nutné na každý požadavek otevřít nové spojení ale naopak lze postupně zaslat více požadavků za sebou, což snižuje režii nutnou k přenosu. [27]

  Udržované spojení s sebou přineslo problém v podobě délky zasílaných dat. V předchozích verzích bylo možné poznat konec souboru podle ukončení spojení. [31] Od verze 1.1 je povinné deklarovat hlavičku `Content-Length`. Aby bylo možné dynamicky generovat stránky s předem neznámou délkou, přibyla možnost data odesílat po blocích, ovšem u nich již musí být uvedena jejich délka. Tento typ přenosu indikuje hlavička `Transfer-Encoding: chunked`. [27]

  Další důležitou změnou je podpora virtuálních WWW serverů. To znamená, že na jedné IP adrese (respektive na jednom serveru) může běžet více WWW serverů. Aby bylo možné odlišit na který z nich je dotaz směřován, je nutné uvést hlavičku `Host` s adresou WWW serveru. [27]

  Mimo to přibyly další metody CONNECT, DELETE, OPTIONS, PUT a TRACE. Jedním z dalších rozšíření je oficiální podpora jazykových mutací nebo přenosu pouze části souboru. [31]

- **HTML/1.2**
  Vylepšením verze 1.1 vznikla nová verze HTML/1.2, ovšem příliš se neuchytila. Došlo k vylepšení rozložení zatížení spoje a ověřování autentizace přístupu. Byla taktéž přidána nova sada hlaviček.[31]

- **HTML/2.0**
  Nová revize HTTP, která se začíná postupně rozšiřovat. Specifikace byla zveřejněna v květnu 2015 jako RFC 7540 a dnes (duben 2018) ji podle serveru W3Techs využívá 24.7% stránek.[40] HTML/2.0 přináší několik výhod. Tou hlavní by měla být podpora *multiplexingu*, tedy přenosu více souborů během jednoho spojení. To je sice možné už od verze 1.0 (ve verzi 1.1 je to dokonce implicitní), ovšem v této verzi došlo ke značnému vylepšení. Tím je možnost určit prioritu jednotlivým dotazům, díky čemuž složitější dotazy (např. velké obrázky) nezpomalí načítání celé stránky. Dále přibila možnost aby server k odpovědi připojil i další s obsahem související soubory (skripty, styly, obrázky), tudíž ve chvíli, kdy je prohlížeč bude vyžadovat, tak je už mít bude načtené. Zatímco v předchozích bylo šifrování volitelné, v této revizi je již výchozí Mimo to přibila komprese hlaviček a lepší podpora přenosu binárních dat.[31]

2.7.3 Cookies

Jedná se o malé množství dat (v rozsahu jednotek kB) které server zašle klientovy v rámci odpovědi. Klientský prohlížeč si tyto data uchová a při každé další komunikaci je zasílá

serveru jako obsah HTTP hlavičky. Toto chování není zaručeno – uživatel může svému prohlížeči zakázat ukládat cookies, může je vymazat, popř. jednodušší prohlížeče cookies vůbec nemusí podporovat. [25]

Cookies slouží k rozlišení jednotlivých uživatelé webové stránky či k uchování jejich nastavení a předvoleb. Tak lze např. zajistit, že uživatel zůstane přihlášen do systému, popř. v internetovém obchodě slouží k uložení nákupního košíku. [25]

Funkci cookies lze doplnit tzv. *WebStorage*. Jedná se taktéž o lokální úložiště pro uložení libovolných dat ve tvaru klíč - hodnota. Hlavní rozdíl oproti cookies je v tom, že tyto data nejsou odesílána v hlavičce na server.

2.7.4 Kódování Base64

Protože původní návrhy sítí počítali pouze s přenosem tisknutelných znaků, může být v určitých situacích problém přenést binární data pomocí některého ze starších protokolů. Proto bylo potřeba vytvořit mechanismus, který převede binární data na posloupnost tisknutelných znaků. Právě k tomuto účelu vzniklo kódování Base64 (a také méně používané Base16 a Base32). [23]

Vstupní binární data jsou kódovány tak, že se každé 3 byty uložení jako 4 tisknutelné ASCII znaky za pomoci následujícího algoritmu a 64-prvkové kódovací tabulky: [23]

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789
```

Algoritmus kódování:

1. Načteme první 3 byty (24 bitů) k zakódování
2. Těchto 24 bitů rozdělíme na 4 skupiny po 6 bitech
3. Získáme 4 čísla v rozsahu 0 – 64 která jsou indexem do kódovací tabulky.
4. Přečteme ASCII znak na daném indexu v kódovací tabulce
5. Opakujeme dokud jsou data na vstupu

Binární data	1 0 1 1 0 1 1 0	1 1 0 0 1 1 1 0	0 0 0 1 1 1 0 0	0 0 0 0 0 1 0 1	
Rozdělení	1 0 1 1 0 1	1 0 1 1 0 0	1 1 1 0 0 0	0 1 1 1 0 0	0 0 0 0 0 1	0 1 0 0 0 0
Index	45	44	56	28	1	16
Znak z tabulky	t	s	4	c	B	Q
Kódovaná data	ts4cBQ==					

Obrázek 2.21: Princip kódování Base64

V případě, že není vstup dělitelný 3, doplní se na konec nuly a do výsledného řetězce se doplní znak `=` tak, aby výsledný počet znaků byl dělitelný čtyřmi. Následné dekódování probíhá přesně opačně. [23]

2.7.5 Odeslání dat na server

Z předchozího popisu se může zdát, že k odeslání dat existuje pouze jedna možnost – metoda POST. Ovšem to není zcela přesné. V dnešní době se využívají v podstatě 3 možnosti, jak serveru předat určité informace.

Metoda POST je přímo určena k odesílání dat. Je podporován od HTTP/1.0, což dnes splňuje drtivá většina serverů. Data jsou umístěna v těle HTTP dotazu, takže je možné odesílání libovolných dat, včetně struktur či binárních souborů.

Metoda GET je sice určena pro získání dat, ale přesto ji lze určitým způsobem využít i pro jejich odesílání. Data jsou vloženy přímo do URL[7] adresy jako její parametry. Od adresy jsou odděleny otazníkem a jsou psány ve tvaru `promenna=hodnota`. Je-li jich potřeba zaslat víc, tak se jako oddělovač používá `&`. [10] Příkladem může být následující adresa:

```
http://www.mujweb.cz/index.php?sekce=clanky&id=456186312
```

Tento způsob je funkční na všech server bez ohledu na verzi HTTP, ale má několik nevýhod vycházejících především z limitů samotné URL adresy.

Velikost přenášených dat HTTP/1.1 sice nedefinuje žádné konkrétní omezení na maximální délku URL [19], ale jednotlivé internetové prohlížeče už ano. Např. Internet Explorer od firmy Microsoft má limit 2083 znaků [5]. Část tohoto limitu zabere samotná adresa, tudíž limit přenesených dat metodou GET je cca 2 kB (včetně jmen proměnných).

Typ dat URL adresa může obsahovat pouze tisknutelné znaky, to znamená, že jakékoli jiné znaky je nutné nějakým způsobem zakódovat. Mimo to problémem jsou i některé speciální znaky jako `=`, `&`, aj. Z toho plyne, že přenos jiných dat než čísla či řetězce je poměrně problémové a je mnohem vhodnější použít metodu POST.

Viditelnost dat Jelikož data jsou součástí URL adresy, jsou viditelné v adresním řádku prohlížeče. To v určitých případech může být výhoda (lze tak uložit data přímo do samotného hypertextového odkazu), ale na druhou stranu má uživatel možnost tyto data snadno upravit, co může vést k neočekávanému chování serveru (pokud s touto situací nepočítal). Je to problém i z bezpečnostního hlediska, např. přenos hesla.

Metoda GET jinak V posledních letech se využívá ještě jeden způsob taktéž založený na metodě GET. Ovšem data nejsou připojena za URL adresu jakožto parametry, nýbrž jsou přímo součástí URL adresy. Příkladem může být adresa:

```
http://www.mujweb.cz/clanky/uvodni-clanek
```

Tato adresa je na serveru buď přímo zpracována nebo zpět přeložena na adresu s parametry:

```
http://www.mujweb.cz/index.php?sekce=clanky&jmeno=uvodni-clanek
```

Pro tento způsob platí všechny výhody i nevýhody zmíněné v předchozím odstavci o metodě GET. Hlavní výhodou tohoto způsobu předávání dat a důvodem, proč je v poslední době v takové oblibě je možnost vytváření tzv. „hezkých“ odkazů. Ty jsou poměrně krátké a dobře vystihují, co adresují.

[7] URL je zkratkou pro Uniform Resource Locator (česky: *Jednotná adresa zdroje*). Jedná se o řetězec znaků definující doménovou adresu serveru, umístění zdroje a protokol. [31]

2.7.6 WWW a související technologie

World Wide Web zkráceně jen *WWW* či *web* je nejznámější služba běžící na Internetu, což je celosvětová síť propojující počítače. Slouží k prohlížení, ukládání a sdílení dat či dokumentů. K tomu využívá právě protokol HTTP.

HTML

Základem webu jsou *html* soubory psané ve jazyce HTML (HyperText Markup Language), který popisuje strukturu (definuje jednotlivé části jako např. hlavička, menu nebo třeba článek) a obsah webové stránky. Jazyk se postupně vyvíjel až aktuální verzi HTML5. Ta na rozdíl od předchozí verze více cílí na strukturování obsahu a naopak ruší příkazy jeho formátování. HTML soubor je obyčejný textový soubor doplněný o tzv. *tagy*, které dávají textu určitý význam. [18]

Kaskádové styly

Kaskádové styly (*Cascading Style Sheets*), dnes ve verzi CSS3, doplňují HTML o popis vzhledu. Tedy zatímco jazykem HTML definujeme např. obsah článku, tak CSS slouží pro popis jeho vzhledu. Toto rozdělení je důležité a všechny moderní webové stránky by ho měli dodržovat. Pomocí CSS3 je možné ovlivnit vzhled prakticky všech elementů na stránce. Navíc má podporu i pro jednoduché animace a pravidla formátování pro různá zařízení (např. mobilní vzhled stránek). Nutno dodat, že ne všechny prohlížeče implementují CSS3 stejně. [18]

Zajímavým rozšířením jazyka CSS o další funkce jsou tzv. CSS preprocesory. Fungují obdobně jako preprocesor pro programovací jazyky. Pomocí něj lze definovat různá makra a proměnné, které se ještě před překladem (v tomto případě před použitím stylu v prohlížeči) přeloží do výsledného souboru. Nejedná se o oficiální rozšíření CSS, což znamená, že je nepodporují ani webové prohlížeče. Překlad musí nějakým způsobem (před nahráním, automaticky na serveru nebo u uživatele v prohlížeči) zajistit programátor, což je cena za jednodušší a efektivnější tvoření kaskádových stylů.

JavaScript a jQuery

JavaScript slouží k definování chování webové stránky. Pomocí něj tedy lze webovou stránku „oživit“ a zajistit tak, aby nějakým způsobem reagovala na podněty od uživatele. Některé z jeho možností postupně přebraly kaskádové styly (animace elementů) či jazyk HTML (validace dat ve formulářích). Ale ve spoustě situací se bez JavaScriptu neobejdeme. Příkladem může být dynamické načítání dat za běhu stránky či změna obsahu elementů na základě určitých podmínek – např. zobrazení aktuálního času. Schopnosti JavaScriptu lze dále rozšířit o další pomocí knihoven. Příkladem může být knihovna *jQuery*, která dnes patří mezi nejrozšířenější. [18]

Zmínil jsem možnost dynamického načítání dat za běhu, tedy bez nutnosti znovu načíst stránku. Tato technologie se nazývá AJAX, přičemž se v podstatě jedná o odeslání požadavku na server a zpracování jeho odpovědi. To vše probíhá na pozadí bez toho, aby o tom uživatel vůbec věděl a tudíž ho to nijak neomezuje v práci. [7]

XML a JSON

XML (*eXtensible Markup Language*) i JSON (*JavaScript Object Notation*) jsou značkovací jazyky, které popisují strukturu dokumentu, ale nespecifikují jeho vzhled. Zatímco XML se stalo základem pro další datové formáty (např. HTML vychází ze XML), tak JSON se naopak prosadil v přenosu strukturovaných dat mezi serverem a uživatelem. [7]

```
<osoba>
       <jmeno>Jan</jmeno>
       <prijmeni>Novak</prijmeni>
</osoba>
```

```
var osoba = {
       jmeno : "Jan",
       prijmeni : "Novak"
}
```

Obrázek 2.22: Jednoduchá struktura popsaná pomocí XML (*vlevo*) a JSON (*vpravo*)

2.8 Existující bezdrátově ovládané osvětlení

Na trhu existuje množství různých řešení pro bezdrátové ovládání osvětlení. Níže uvedu několik příkladů.

2.8.1 LED pásek ovládaný dálkovým ovladačem

Velmi běžným a poměrně dostupným řešením je LED pásek. Díky lepící pásce na spodní straně lze pásek umístit téměř kamkoli a vytvořit tak zajímavě nasvícený interiér. LED pásky jsou poměrně nenápadné, tudíž nepůsobí rušivě. LED pásek je nutné doplnit o řídící jednotku, která může být vybavena nějakým způsobem bezdrátového ovládání. Často se jedná o jednoduchý dálkový ovladač. Toto osvětlení lze buď nakoupit a sestavit po částech nebo se poohlédnout po nějakém setu. V obou případech to ale zpravidla vyžaduje alespoň základní znalosti elektroniky a technickou zručnost.

Obrázek 2.23: Set LED pásku s řídící jednotkou

2.8.2 Chytrá žárovka

Pod tímto označením si lze představit ledasco, čehož občas zneužívají prodejci. Ono i označení žárovka není zcela přesné, protože se zpravidla jedná o světlo na bázi LED.

Instalace je jednoduchá, většinou stačí pouze vyměnit současnou žárovku za novou, spustit aplikaci v mobilu a je hotovo. Některé žárovky mohou pro svoji činnost vyžadovat ještě centrální řídící zařízení, k němuž se uživatel připojuje skrze aplikaci a až ono komunikuje s jednotlivými žárovkami.

Například žárovky *Philips Hue* z obrázku 2.24 komunikují s uživatelem pomocí Wi-Fi. Respektive uživatel komunikuje s centrálním zařízením *Hue Bridge* (na obrázku se nachází vlevo), které pak řídí jednotlivé žárovky. To umožňuje snadno propojit více žárovek v jeden celek a vytvářet tak různá pravidla (tlumené osvětlení pro sledování TV, ranní budíček pomocí světel, apod.)

Obrázek 2.24: Chytré žárovky značky Philips

Jiným způsobem je přímá komunikace s uživatelem bez centrální jednotky a to buď na principu Bluetooth nebo Wi-Fi. Občas lze také narazit na ovládání pomocí dálkového ovladače jako u LED pásků.

Kapitola 3

Zhodnocení současného stavu a plán práce

Současná nabídka systémů pro ovládání osvětlení, je poměrně široká a nabízí nepřeberné množství kombinací. V následující podkapitole zmiňuji některé možné principy bezdrátového ovládání včetně výhod i nevýhod spojených s jejich používáním. Po té se zaměřím na možné kompromisy spojené s používáním existujících řešení. Na základě těchto faktů v závěru definuji několik vlastností, kterých bych chtěl u výsledného zařízení dosáhnout.

3.1 Možnosti ovládání osvětlení

K ovládání osvětlení se nabízí velká škála ovládacích prvků. Některé se běžně používají, jiné jen ojediněle. Každý má své výhody i nevýhody a záleží tak především na uživateli, kterým dá přednost.

Zvuk Možnost rozsvítit světlo např. písknutím nebo lusknutím prstů se prakticky nikde nepoužívá, i když se jedná o technicky snadno proveditelné a levné řešení. Jako nevýhody lze zmínit náchylnost na rušení okolím a i samotné rušení okolí – nutnost vydávat zvuk by mohla někoho obtěžovat. Jako výhodu lze naopak zmínit možnost snadného „bezdrátového“ ovládání které nepotřebuje žádné speciální ovládací zařízení.

Dálkový ovladač Na trhu lze potkat spoustu osvětlení využívajících tento typ ovládání. Z osobní zkušenosti ovšem musím podotknout, že nutnost použít dálkový ovladač je vcelku nepohodlná. Zároveň je komunikace omezena pouze na jeden směr (od uživatele do zařízení) což může být určitá nevýhoda.

Bluetooth je označení pro standart bezdrátové komunikace mezi zařízeními. Existuje v několika verzích (dnes ve verzi 5.1) a je tvořen sadou protokolů pro přenos dat, hudby nebo třeba internetového připojení. Většina telefonů vyrobená v posledních letech je jím vybavena, tudíž se jeho použití přímo nabízí. Bohužel je to podmíněno speciální aplikací, jelikož Bluetooth neobsahuje protokol pro ovládání osvětlení.

Wi-Fi podobně jako Bluetooth, označuje sadu standardů popisující bezdrátovou komunikaci v počítačových sítích. S Bluetooth má společnou i rozsáhlou rozšířenost. I v tomto případě se většinou neobejdeme bez speciální aplikace.

3.2 Seznam některých kompromisů současných řešení

Jak jsem zmínil výše, některé z možností bezdrátového ovládání se moc nepoužívají (ovládání zvukem), ale jiné se poměrně rozšířili. Většina dnešních bezdrátově ovládaných osvětlení využívá buď bezdrátový ovladač, Bluetooth nebo Wi-Fi. Bohužel občas je jejich používání podmíněno určitými kompromisy. Rád bych zmínil některé z nich včetně příkladu a možnosti jeho řešení.

3.2.1 Nutnost výměny současného osvětlení

Většina dnes nabízených řešení je ve formě led žárovky, popř. led pásku i s ovládací stanicí. Na jednu stranu odpadá složitá montáž, jelikož uživateli praktický stačí jen vyměnit žárovku. Na druhou stranu je uživatel nucen vyřadit své současné osvětlení. Řešením by bylo nabízet pouze samotnou ovládací stanici. Toho si je sice několik výrobců vědomo a takové zařízení nabízí, ale často se jedná spíše o výjimku.

3.2.2 Nefunkční aplikace

Tento neduh je doménou především levnějších zařízení (přičemž se nemusí jednat přímo o osvětlení), kdy výrobce při snaze minimalizovat náklady omezí financování vývoje aplikace. V důsledku toho je plná chyb, dochází k neočekávaným pádům, apod. Problém je, že ze strany uživatele má tato situace jediné řešení: na základě recenzí se danému výrobku buď vyhnout nebo doufat, že výrobce chyby postupně odstraní. Bohužel zejména u starších výrobků se vývoj aplikace často po určité době navždy zastaví...

3.2.3 Uzavřená komunikace

S předchozím bodem lehce souvisí i uzavřenost komunikace. Málokterý výrobce zveřejňuje popis komunikace mezi aplikací a osvětlením. Tato vlastnost komplikuje integraci do vlastního systému osvětlení, popř. komunitní vývoj vlastní ovládací aplikace. Zveřejnění komunikace by sice výrobci nijak významně nezvýšilo náklady, ale na druhou stranu by mohlo odhalit slabiny systému a mohlo by být zneužito k útoku, což bude nejspíš hlavní důvod, proč to téměř nikdo nedělá.

3.2.4 Omezená možnost integrace

Někteří výrobci sice podporují integraci svých výrobků do systémů chytré domácnosti, jako je například *Google Home* nebo *HomeKit* od Applu, ale rozhodně se nejedná o pravidlo. Což přináší další problém v podobě nutnosti mít několik aplikací (pro každého výrobce jednu) a nemožnost jednotlivá zařízení mezi sebou propojit.

3.2.5 Omezení na několik platforem

Výrobce často ke svému svítidlu vydá aplikaci pouze pro Android a iOS. Málokdy se setkáme s aplikací pro Windows Mobile. Stolní počítače, notebooky či „hloupé“ telefony pak mívají zpravidla smůlu a k ovládání osvětlení je nelze použít. Řešením by byla existence univerzálního protokolu pro ovládání osvětlení a jeho integrace do operačních systému zařízení, ale o tom v současnosti zatím nikdo neuvažuje.

3.2.6 Závislost na internetovém připojení

Občas můžeme narazit na možnost ovládat osvětlení i vzdáleně přes Internet. Tato vlastnost může být zajímavá i užitečná. Problémem je, pokud se jedná o jedinou variantu. Osvětlení je pak totiž zcela závislé na internetovém připojení ale hlavně na funkčnosti serverů výrobce. A jakýkoli výpadek zapříčiní nemožnost ovládat vlastní osvětlení. Proto by se mělo jednat pouze o doplňkovou funkci a stále by mělo být dostupné ovládání po lokální síti.

3.3 Upřesnění zadání

V předchozí podkapitole jsem zmínil několik neduhů, kterými trpí současná řešení. A protože bych byl rád, aby jimi výsledné zařízení netrpělo, definoval jsem následující seznam požadovaných vlastností.

Použitelnost se současným osvětlením Tedy výsledkem by nebyl přímo zdroj světla doplněný o ovládací modul, nýbrž pouze zmíněný modul se sadou výstupů, na které by bylo možné připojit současné osvětlení. Z toho vychází i následující bod.

Podpora různých typů osvětlení V domácnosti zpravidla nenajdeme jeden typ osvětlení. Tuto vlastnost by zařízení mělo respektovat, proto jsem se rozhodl podporovat stmívatelné i nestmívatelné osvětlení včetně RGB osvětlení.

Použití Wi-Fi K použití Wi-Fi jsem se rozhodl především proto, že mám osobní zkušenosti s ovládáním pomocí dálkového ovladače, přičemž takové ovládání není vůbec komfortní. Sice se nabízí ještě Bluetooth, ale ten by si vyžádal existenci speciální aplikace, tudíž by nastal konflikt s následujícím bodem.

S použitím Wi-Fi dále přibývá podmínka, že zařízení nesmí být závislé na existenci domácí Wi-Fi sítě a internetovém připojení. Obě tyto podmínky sice mírně komplikují návrh i vývoj, ale jak už jsem zmínil, pro koncového uživatele se může jednat o značnou komplikaci v případě výpadku.

Multiplatformnost Zařízení by mělo být možné ovládat z co největšího počtu různých systémů. Moderní programovací jazyky jsou sice multiplatformní včetně možnosti fungování na stolních počítačích i mobilních zařízeních, ale vždy se najde platforma, na které nefungují nebo fungují špatně. Proto jsem se rozhodl uživatelské prostředí vytvořit jako webovou stránku. Takto lze k ovládání použít jakékoli zařízení obsahující internetový prohlížeč a Wi-Fi modul bez ohledu na operační systém nebo jeho verzi.

Otevřená komunikace Aby si pokročilý uživatel mohl navrhnout vlastní řídící aplikaci nebo jen integrovat zařízení do vlastního systému, měl by mít přístup k popisu komunikace. Proto by měl společně s prototypem vzniknout i seznam všech příkazů včetně jejich detailního popisu.

Co nejnižší cena Samozřejmě jsem se snažil o co nejnižší výsledné náklady.

Mám-li shrnout předchozí řádky, tak současný trh nabízí dostatečně širokou nabídku osvětlení. Přesto může uživatel narazit na některé možné kompromisy a určitě tak má smysl vytvořit další alternativu.

Kapitola 4

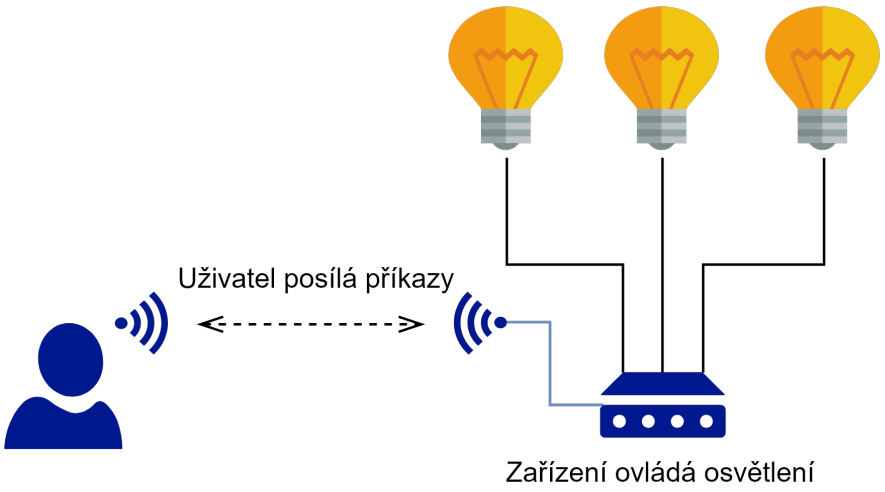
Vývoj zařízení

Vývoj zařízení začíná nejdříve obecným návrhem. Ten se postupně konkretizuje až je dostatečně podrobný a je možné pokročit k samotnému vývoji. V závěru přichází na řadu výroba prototypu a testování jeho jednotlivých částí. Těmto jednotlivým krokům se věnují následující podkapitoly.

4.1 Základní návrh a výběr komponent

Funkci zařízení popisuje blokové schéma na obrázku 4.1. Cílem je umožnit uživateli pomocí libovolného zařízení s Wi-Fi kontrolovat aktuální stav osvětlení a řídit ho – mít možnost ho zapnout, vypnout či změnit jas nebo barvu.

To znamená, že řídící zařízení musí být také vybaveno Wi-Fi modulem. Mimo to musí obsahovat paměť pro uchování nastavení a sadů výstupů pro připojení ovládaných světelných zdrojů. Dále je třeba vhodně navrhnout schéma komunikace mezi zařízením a uživatelem, respektive uživatelským prostředím, které bude nutné navrhnout.



Obrázek 4.1: Blokové schéma popisující funkci zařízení

Abych toho docílil, je nutné projít několik kroků:

- Vybrat vhodné komponenty
- Zjistit jak spolu komunikují a navrhnout jejich zapojení
- Navrhnout pravidla pro komunikaci
- S tím souvisí návrh uživatelského rozhraní
- Důležitým a opomíjeným krokem je testovaní – tedy co se bude testovat a jakým způsobem se to bude testovat (podrobněji podkapitola 4.7)

4.1.1 Výběr komponent

Výběr bude vhodné začít u Wi-Fi modulu. Na trhu se jich nachází několik, ale mě nejvíce zaujal ESP-01 (občas také označován jako ESP8266-01). Cena tohoto modulu začíná na 1,5 $, přičemž se chlubí velmi zajímavými parametry:

- 802.11 b/g/n protokol
- Možnost vytvořit vlastní síť nebo se připojit k existující
- Podpora TCP/IP
- Komunikace pomocí UART

Na tento Wi-Fi modul je možné nahrát program a je tak zcela samostatný, limitem je ale omezený počet pinů, konkrétně 3. Proto jsem jej doplnit o MCU s větším počtem pinů. Vybral jsem si dnes velmi populární Arduino – vývojová platforma je pro návrh prototypu vhodnější. Konkrétně jsem zvolil Arduino Nano, zejména z důvodu, že je poměrně levné (respektive jeho klony) a malé. Dalším benefitem je existence širokého množství knihoven a návodů. Bylo by samozřejmě možné použít i jakýkoli jiný MCU s UART modulem.

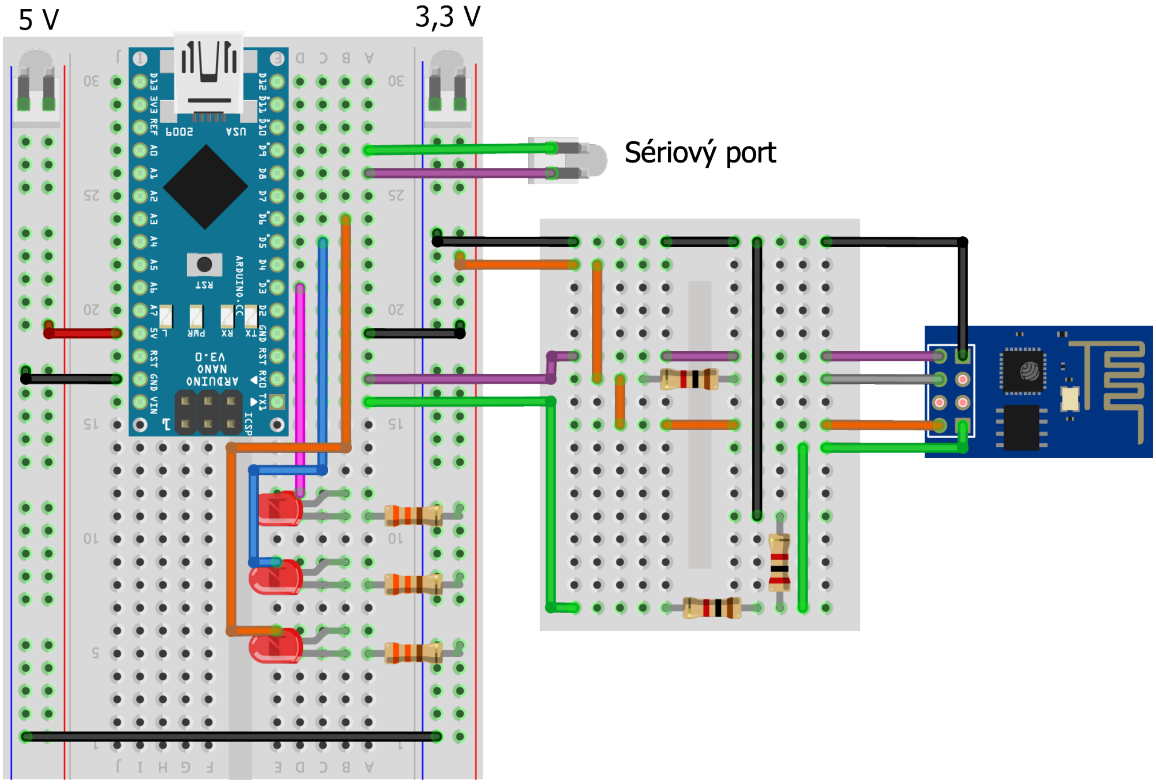
Arduino Nano je vybaveno MCU ATmega328. Ten nabízí 32 KiB FLASH paměti, 2 KiB RAM paměti a 1 KiB EEPROM paměti pro uchování uživatelského nastavení. K dispozici je 22 pinů (z toho 6 s podporou PWM), což pro prototyp postačuje, ale v případě ostrého nasazení by bylo vhodné tento počet rozšířit. A to buď použitím jiného MCU s větším počtem pinů nebo pomocí tzv. *expandéru*. Například integrovaný obvod TLC5940 je schopný generovat až 16 různých PWM signálů a umožňuje řetězení, tedy jich lze použít více za sebou a počet generovaných signálů dále zvyšovat. Další součástky nejsou pro vývoj prvotního prototypu nutné. Pozdějšímu rozšíření se věnuji v podkapitole 4.6.

Prvotní prototyp		
MCU	Arduino Nano	1×
Wi-Fi	ESP8266-01	1×
Rezistor	330 Ω	3×
	1 kΩ	2×
	2 kΩ	1×
LED	Indikační	3×

Tabulka 4.1: Seznam potřebných součástek

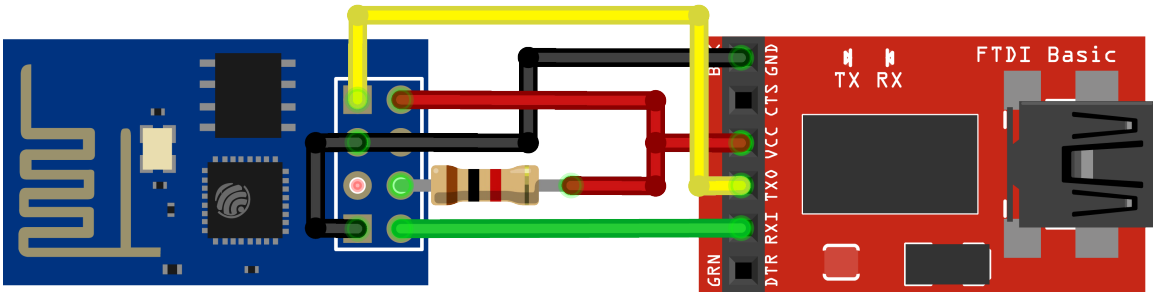
4.1.2 Schéma zapojení

Na obrázku 4.2 je znázorněné základní zapojení zařízení (schéma se nachází v příloze na obrázku A.1) v nepájivém poli. Je zde vidět několik kolizí, které je nutné řešit.



Obrázek 4.2: Základní zapojení

Mimo to bude nutné do modulu nahrát alternativní firmware. Propojení ESP-01 se sériovým portem PC (postačí i převodník) je znázorněno na obrázku 4.3 (schématicky v příloze na obr. A.2). Důležité je uzemnění pinu `GPIO_0` a dodržení maximálního napětí 3,3 V. Podrobnosti zmiňuji dále v podkapitole 4.2.1 věnující se podrobněji Wi-Fi modulu.



Obrázek 4.3: Zapojení pro nahrání firmware

Wi-Fi modul pracuje s napájecím napětím 3,3 V. Toto napětí také používá pro logickou jedničku, zatímco Arduino je 5V, tudíž by hrozilo přetížení obvodů na straně Wi-Fi modulu. Ale to není důvodem pro změnu MCU. ATmega328, kterou používá Arduino, akceptuje i

napájecí napětí 3,3 V. Pouze by to znamenalo hardwarovou úpravu na desce Arduina a to výměnu stabilizátoru napětí.

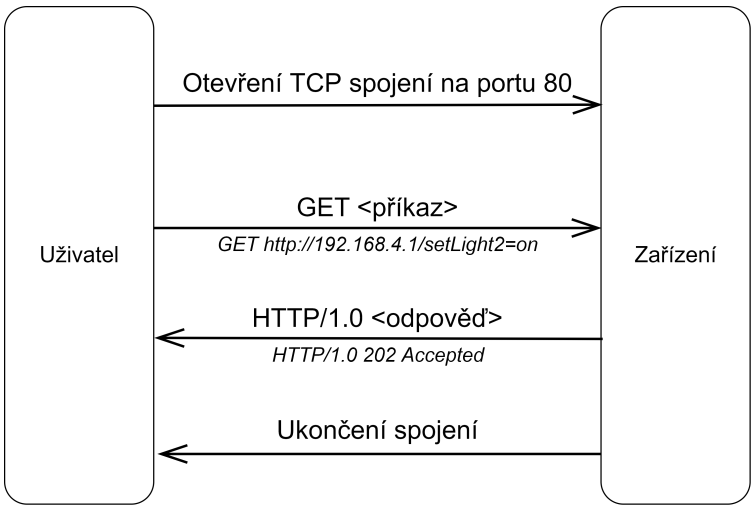
Protože po ukončení vývoje prototypu by bylo vhodnější zcela vynechat vývojovou desku Arduino a místo ní přímo použít MCU, tak jsem se rozhodl to vyřešit jinak a zajistil jsem pro Wi-Fi modul vlastní zdroj napětí. Teoreticky by sice bylo možné využít přímo 3,3V pin na desce Arduina, ale prakticky je to zcela nevhodné. Tento pin je napájen integrovaným obvodem FT232RL který se stará o převod USB – sériový port a maximální povolené zatížení tohoto pinu je 50 mA. [3]

Mimo napájení je také nutné snížit napětí logické jedničky z Arduina. Toho lze snadno docílit pomocí napěťového děliče složeného z rezistorů s hodnotou 1 kΩ a 2 kΩ. Opačným směrem by naopak bylo vhodné napětí zvýšit, např. pomocí NPN zesilovače, ale následné testování a zkušenosti ostatních ukazují, že to není vyloženě nutné.

Arduino je vybaveno pouze jedním sériovým portem, který využívám pro komunikaci s Wi-Fi modulem. To způsobuje problémy při nahrávání programu, kdy je nutné zabránit Wi-Fi modulu zasahovat do komunikace mezi PC a ATmegou. Druhým problémem je testování a ladění. Arduino samo o sobě nemá přímou podporu pro krokování programu. Je nutné proto nějakým způsobem zajistit přístup k datům v paměti. Např. pomocí ladících výpisů odesílaných na sériový port PC, k čemuž je potřeba druhý UART modul. Ten lze softwarově emulovat za pomoci oficiální knihovny `SoftwareSerial.h`. Takto vytvořený port pracuje zcela stejně jako hardwarový a jediný (i když vcelku důležitý) rozdíl mezi nimi je ve větší zátěži CPU, která se musí starat o celou komunikaci, zatímco v druhém případě se o to postará modul. Na schématu je tento výstup vyveden na piny označené *Sériový port*.

4.1.3 Návrh pravidel pro komunikaci

Prvotní návrh využívá pro odesílání dat metodu GET, přičemž příkazy jsou kódovány přímo do URL adresy z důvodu čitelnosti a nižších nároků na zpracování. MCU po zpracování příkazu odesílá HTTP/1.0 odpověď *200 OK*, bylo-li třeba odeslat nějaká data. Častější je spíše *202 Accepted*, což značí, že dotaz byl úspěšný, ale nejsou zasílána žádná data v těle zprávy. Chyby jsou hlášeny chybovými kódy *4xx*. Na obrázku 4.4 je znázorněn základní princip komunikace. Tomuto tématu se podrobněji věnuji v podkapitole 4.3.2.



Obrázek 4.4: Princip komunikace včetně ukázky příkazu pro rozsvícení světla s ID 2

4.1.4 Návrh uživatelského rozhraní

V mém případě jsem kladl důraz na dotykové ovládání a adaptaci rozhraní na mobilní obrazovky. Ale zároveň bylo třeba zachovat možnost ovládání pomocí myši na velké obrazovce počítače. S ohledem na to, že komunikace se zařízením probíhá po síti, rozhodl jsem se uživatelské prostředí vytvořit ve formě webové stránky. Díky tomu nevzniká omezení na konkrétní platformu, jedinou podmínkou je přítomnost webového prohlížeče.

Každé jedno světlo jsem se rozhodl reprezentovat jako samostatný blok, který bude zobrazovat dostupné informace (zda svítí, aktuální jas, apod.), bude umožňovat změnit jeho stav a nastavení. Přibližnou podobu tohoto bloku znázorňuje nákres 4.5. Podrobnější popis se nachází v podkapitole 4.5.

Obrázek 4.5: Blok pro ovládání jednoho světla

4.2 Wi-Fi modul – komunikace s ním, knihovna pro Arduino

Zprovoznit Arduino platformu na PC s operačním systémem Windows nepředstavuje žádný problém. To stejné se dá říct i o Wi-Fi modulu, který je už z výroby připraven na zapojení. Abych byl konkrétnější, při propojení modulu s Arduinem můžou nastat určité komplikace. Wi-Fi modul ve výchozím nastavení komunikuje rychlostí 115 200 Bd, zatímco sériová linka ATmegy328 není schopna komunikovat rychleji než 9 600 Bd. Protože některé Wi-Fi moduly přímo z výroby neumožňují měnit rychlost, je nutné do něj nahrát alternativní firmware.

4.2.1 Nahrání firmware

Před samotným nahráním je potřeba vybrat vhodný firmware. Těch existuje hned několik v několika verzích. Já jsem zvolil *nonOS SDK* verze 2.1.0, který jsem nahrál pomocí programu *NodeMCU Flasher*. Použitý program i firmware se nachází na přiloženém CD v podsložce `ESP8266`.

Postupoval jsem podle návodu z internetové stránky [20] a informací z datasheetu [6]. K nahrání je potřeba připojit ESP-01 k sériovému portu PC podle schématu 4.3. Program *Flasher* je poměrně přehledný. Konkrétní nastavení se odvíjí od zvoleného firmware a velikosti FLASH paměti na modulu. Mnou použité nastavení se nachází v příloze B.

Po dokončení nahrávání je na řade ověření úspěšnosti této operace. Nejdříve je nutné odpojit uzemnění pinu `GPIO_0` a restartovat Wi-Fi modul (například krátkým odpojením napájení). Nyní je třeba na PC spustit libovolný program schopný komunikace pomocí sériové linky. Výchozí rychlost komunikace je 115 200 Bd. Všechny odesílané příkazy musí být zakončeny pomocí `CR LF`. Stejné ukončení používá modul i u odpovědí. Odesláním řetězce `AT` dochází k odeslání tzv. testovacího příkazu. Pokud je vše správně, modul by měl odpovědět `OK`. Odeslání `AT+GMR` vyžádá informace o aktuálním firmware. Podle odpovědi si lze zkontrolovat, zda verze odpovídá té, která byla nahrávána. Na obrázku 4.6 je ukázka komunikace z testování funkčnosti modulu.

```
AT
OK

AT+GMR
AT version:1.4.0.0(May 5 2017 16:10:59)
SDK version:2.1.0(116b762)
compile time:May 5 2017 16:37:48
OK
```

Obrázek 4.6: Ukázka komunikace s Wi-Fi modulem

4.2.2 AT příkazy

Příkazy uvedené výše se souhrnně nazývají *AT příkazy*. Ty jsou vždy tvořeny písmeny `AT+` následovány názvem příkazu, popř. jeho parametry. Jejich kompletní popis lze najít v dokumentaci [6]. K dispozici jsou 4 základní tvary:

AT+<x>=? získá seznam parametrů a jejich rozsah které lze použít.

AT+<x>? zobrazí aktuální hodnotu.

AT+<x>=<...> přečte uživatelem definované parametry z příkazu a vykoná ho.

AT+<x> spustí příkaz bez parametrů.

Každý příkaz musí být ukončen pomocí `CR LF`. Následuje odpověď zakončená `OK` (některé příkazy neposílají odpověď, ale pouze `OK`) popř. chybou (`ERROR`) a opět `CR LF`.

Vrátím se k tomu, proč bylo nutné nahrát jiný firmware – tedy rychlost komunikace. Nový firmware obsahuje AT příkaz `AT+UART` který umožňuje nastavovat vlastnosti komunikace jako rychlost, počet bitů, paritu, atd. Pro provoz s Arduinem jsou vhodné následující parametry:

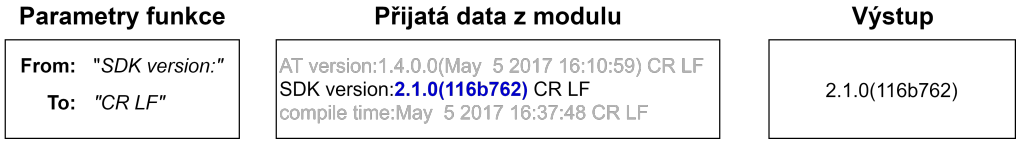
```
AT+UART=9600,8,1,0,0
```

Ve výchozím nastavení je každý modulem přijatý znak zároveň odeslán zpět na terminál uživatele. To je vhodné v případě, že s modulem komunikuje člověk, jelikož některé terminály nezobrazují odeslané znaky a modul tak simuluje zpětnou vazbu z klávesnice. Naopak v případě, kdy s modulem komunikuje jiné zařízení se jedná o zbytečnou zátěž sériové linky. Modul umožňuje toto chování ovlivnit a to pomocí příkazu `ATE0` pro vypnutí či `ATE1` pro zapnutí zrcadlení znaků.

4.2.3 Knihovna pro Arduino

Knihovna pro Arduino funguje na principu odesílání sestavených příkazů na sériovou linku a čtení odpovědí. Mnou testované knihovny vykazovali poměrně vysoké nároky na paměť MCU a proto jsem se rozhodl napsat si vlastní.

Základem celé knihovny je funkce `wifi_readFromTo()`, která se stará o čtení odpovědí od modulu. Čtení probíhá tak, že nejdříve ignoruje všechny příchozí znaky až po první výskyt řetězce *from*. V tu chvíli začne přijaté znaky naopak zaznamenávat. V ideálním případě ukončí čtení přečtením řetězce *to*. Díky tomu, že v dokumentaci modulu je u všech příkazů vždy přesně definovaný tvar odpovědi, lze takto snadno přečíst pouze důležitá data z odpovědi bez nutnosti ji celou uchovávat v paměti. Protože je možné, že čtení z nějakého důvodu selže, je nutné definovat časový limit, po kterém už nemá smysl čekat na odpověď. V rámci prototypu jsem použil 10 s. V reálné aplikaci by bylo vhodnější zvolit nějaký kratší časový limit, např. 3 s.



Obrázek 4.7: Princip funkce `wifi_readFromTo()`

Wi-Fi modul je vybaven vlastní FLASH pamětí pro uchování nastavení, což znamená že není nutné při spuštění vše znovu nastavovat. Pouze stačí zkontrolovat zda modul odpovídá, spustit a nastavit TCP server na portu 80 (port protokolu HTTP) a to je vše.

Knihovna musí podporovat také komunikaci s více klienty naráz, tedy v rámci možností MCU. Wi-Fi modul zvládá obsloužit v jednu chvíli až 4 připojení. Problémem je výše zmíněná funkce, jelikož ignoruje veškerou komunikaci, kterou zrovna neočekává. Tou ale může být i žádost o nové připojení. Postupně jsem dospěl k následujícímu řešení. Ignorovaná komunikace je kontrolována na nové příkazy. Jakmile dojde ke shodě, je tento příkaz zařazen do fronty k obsloužení.

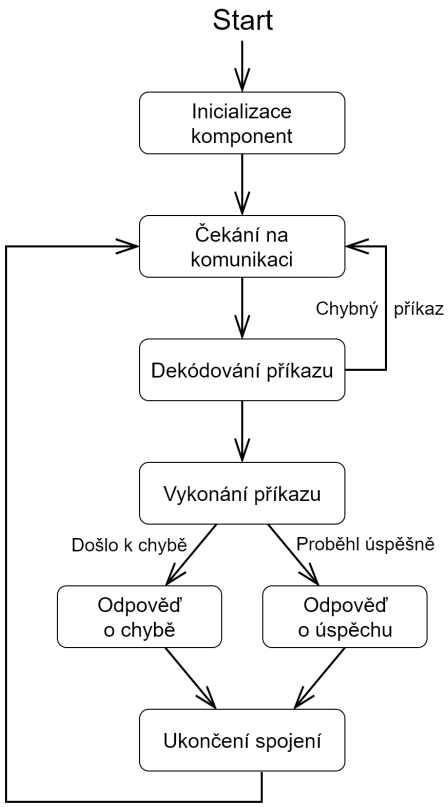
Zajímavou funkcí je `wifi_sendData()`, která se stará o odesílání dat. Jelikož jednou se odesílaná data nachází v RAM jindy třeba zase v EEPROM, tak je pokaždé nutné použít jinou funkci pro čtení. Abych nemusel vytvářet variace funkce `wifi_sendData()` podle toho, ke kterým datům zrovna přistupuje, rozšířil jsem ji o další parametr, který slouží k předání ukazatele na čtecí funkci. To zaručuje i do budoucna dostatečnou univerzálnost a možnosti rozšiřitelnosti. Dobrým příkladem může být odesílání dat v podobě JSON. Stačí vytvořit funkci, která obalí libovolný řetězec JSON strukturou a předat její adresu funkci `wifi_sendData()`.

4.3 Struktura kódu a popis komunikace

Návrh programu je vhodné začít základním popisem, jak má vůbec fungovat a co má dělat. Poté se lze zabývat i tím, jak to bude dělat. S tím lehce souvisí i princip komunikace programu (v tomto případě i celého mikrokontroléru) s okolím.

4.3.1 Základní myšlenka

Základní myšlenku kódu znázorňuje diagram 4.8. Je poměrně jednoduchá. Mikrokontrolér čeká na síťovou komunikaci. Každý příchozí příkaz se pokusí dekódovat. Povede-li se to, přečtou se jeho parametry a dojde ke spuštění. Podle úspěšnosti je vytvořena a odeslána patřičná odpověď a následně po ukončení spojení program opět čeká na další komunikaci.



Obrázek 4.8: Základní myšlenka fungování programu

4.3.2 Komunikace mezi uživatelem a zařízením

Nyní je již vhodný čas na detailní návrh komunikace. Nejdříve je nutné si ujasnit, jaké operace by mohl uživatel chtít provádět se svým osvětlením. Ty jsou znázorněny na obrázku 4.9. Mimo to jsou na něm barevně vyznačeny operace, které spolu blízce souvisí. Například změna jasu a vypnutí světla v obou případech ovlivní jedno konkrétní osvětlení.

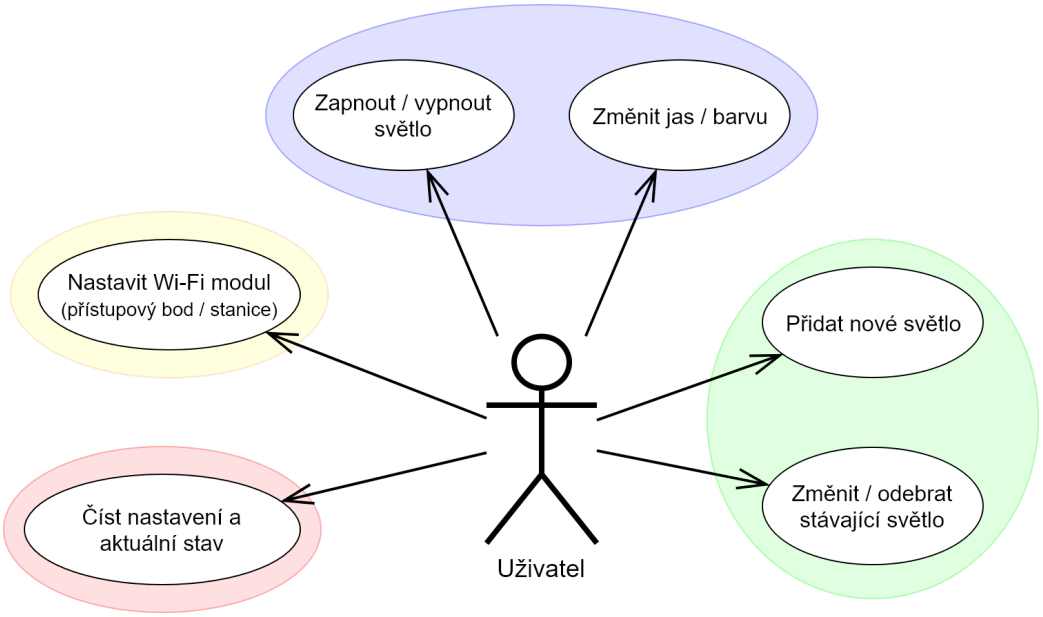
Tyto společné operace by tak mohli být v rámci komunikace kódovány jako jeden příkaz s rozdílnými parametry, tudíž si vystačíme pouze se 4 příkazy. Jejich přesný popis se nachází v příloze C, zjednodušeně řečeno se jedná o následující:

setLight<id>=<stav> nastaví světlo s indexem *id* na *stav*, přičemž stav značí, zda se má zapnout, vypnout nebo změnit jas či barva.

load<id>=<data> načte nové nastavení pro světlo s indexem *id*.

getStatus slouží pro získání aktuálního stavu zařízení.

set<Client/Server>=<nastavení> nastaví Wi-Fi modul.



Obrázek 4.9: Seznam možných operací

Tyto příkazy je nutné dopravit od uživatele přes síť až do zařízení. Je možné si navrhnout buď vlastní protokol, což je velmi efektivní řešení, neboť lze přesně určit jaká data a jakým způsobem budou přenášena. Druhá možnost je použít některý ze stávajících. Abych mohl uživatelské prostředí vytvořit jako webovou stránku, bylo by vhodnější vybrat si druhou variantu a použít protokol HTTP.

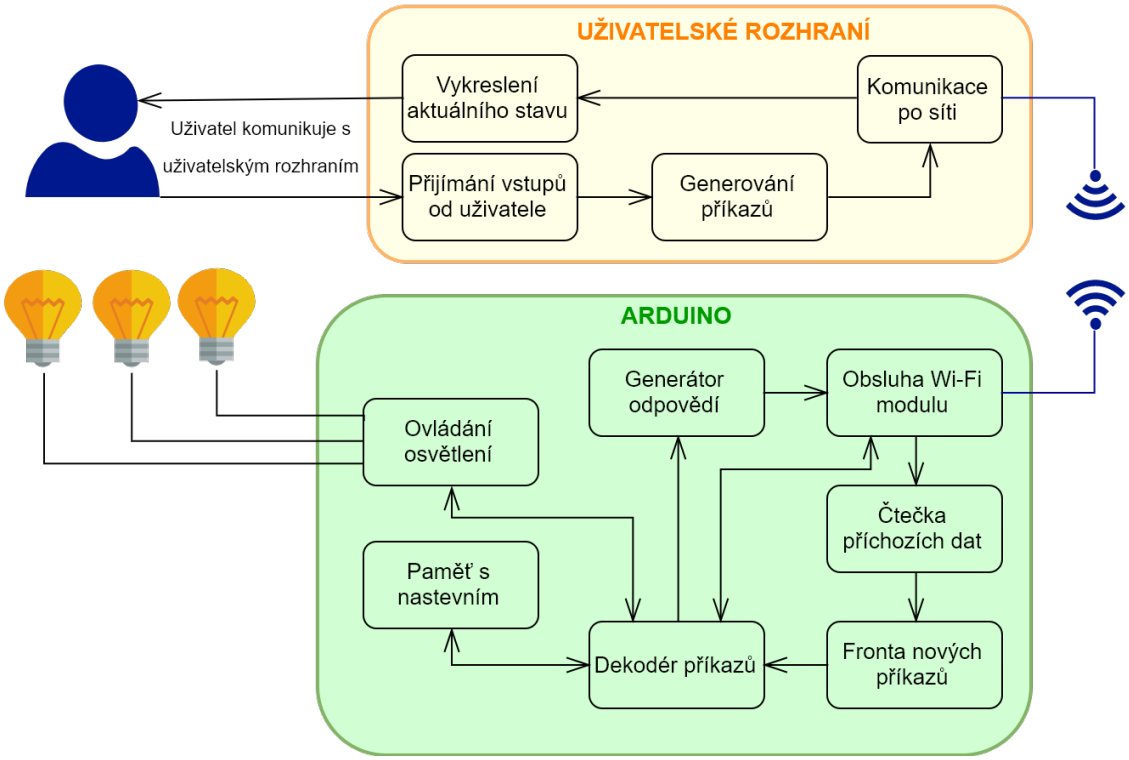
Na přenos dat pomocí HTTP protokolu je určena metoda `POST`. Ale to není jediná cesta, lze také využít metodu `GET`. Ta oproti použití metody `POST` přináší několik výhod. Vygenerovat HTTP hlavičku pro příkaz `GET` je mnohem snazší – postačí napsat libovolnou adresu do adresního řádku prohlížeče. Zpracování na straně MCU je také jednodušší. Veškeré údaje se totiž nachází hned na prvním řádku přijatých dat, tudíž není nutné číst celou přijatou komunikaci.

Při přenosu metodou `GET` jsou data uložena přímo do URL adresy. Buď se připojí na konec jako parametry nebo je možné je přímo zakódovat do samotné adresy. Parametry jsou obecnější a umí je generovat i samotný prohlížeč. Ale pokud by se mělo zařízení chovat standardně, tak by mělo umožnit tímto způsobem odeslat i více různých parametrů a nezáleželo by na jejich pořadí. To je na zpracování samozřejmě obtížnější. Proto jsem dal přednost vytváření specifických URL adres, což mi umožní definovat přesný tvar příkazů.

Odpovědi od zařízení využívají HTML stavové kódy – prohlížeč už na základě kódu je schopný sám rozhodnout, zda byla operace úspěšná (kódy `200 OK` a `202 Accepted`) nebo selhala (`400 Bad Request`).

4.3.3 Struktura kódu

Poté co je definována základní myšlenka programu a jeho komunikace s okolím, je možné navrhnout samotnou strukturu kódu. K tomu jsem využil původní blokové schéma 4.1 z úvodu této kapitoly, které jsem rozšířil o *podbloky* představující logické bloky v rámci výsledného kódu. Toto schéma se nachází na obrázku 4.10.



Obrázek 4.10: Blokové schéma popisující fungování projektu

Činnost uživatelského rozhraní je poměrně jasná. Přijímá vstupy od uživatele. Z nich generuje příkazy které odesílá po síti. Odpovědi ze zařízení jsou dekódovány a graficky znázorněny uživateli.

Blok znázorňující Arduino je již složitější a neobejde se bez vysvětlení. Jak je vidět, komunikace s okolím probíhá pouze pomocí Wi-Fi modulu. Přijímaná data zpracovává *čtečka příchozích dat*, která všechny nové příkazy řadí do *fronty*. Ty jsou dekódovány pomocí *dekodéru*, který se zároveň stará o jejich vykonání v patřičném bloku. Tyto bloky vrací dekodéru odpověď, zda byl příkaz úspěšně vykonán. Tato informace je dále předána *generátoru odpovědí* a výsledná odpověď je odeslána zpět uživateli a poté je ukončeno spojení.

4.4 Samotná implementace a optimalizace

I když je programování MCU do jisté míry stejné jako PC, je nutné brát v potaz určité odlišnosti. Například řádově menší paměť RAM. Zatímco osobní počítače jsou dnes běžně vybaveny 4 GiB paměti, tak použité Arduino má k dispozici pouze 2 KiB paměti, což je dvou-milionkrát méně. Na první pohled je to hodně, tedy spíše hodně málo, ale jen díky tomu mohou být MCU dostatečně malé a levné. Proto jsem musel při návrhu myslet i na budoucí optimalizace kódu. A tím nemyslím optimalizace, které provádí překladač při překladu do strojového kódu, nýbrž ty, které může udělat i programátor.

Pro programování jsem zvolil jazyk C, jakožto kompromis mezi rychlostí programování a hardwarovou náročností výsledného programu. Sice se nabízel jazyk C++, který by o něco málo zrychlil programování a zkrátil kód. Ostatně většina knihoven pro Arduino ho pou-

žívá. Jenže C++ je oproti jazyku C výkonově náročnější a více odstiňuje programátora od hardwaru, což je v případě programování MCU nežádoucí.

Každá funkce je nejdříve deklarována v hlavičkovém souboru, kde je také doplněna o dokumentační komentáře. Díky tomu nástroje, jako je např. *Doxygen*, mohou generovat automatickou dokumentaci pro knihovny. Tu jsem vložil na přiložené CD do složky `Dokumentace`. Více podrobností o struktuře zdrojových souborů se nachází v příloze D.

Dále uvede několik konkrétních případů z implementace, které mi připadají zajímavé nebo důležité.

4.4.1 Základní funkce `main()`

Při programování je zvykem umístit začátek programu do funkce `main()`, z které jsou posléze volány další jednotlivé funkce a procedury. V případě Arduina je to jinak. Místo jedné funkce `main()` jsou tu dvě hlavní funkce – `setup()` a `loop()`.

Funkce **`setup()`** je vstupem do programu. Je volána pouze jednou a to ihned po spuštění MCU. V mém případě obsahuje inicializaci všech potřebných komponent, jako je Wi-Fi modul nebo třeba sériové porty (fyzický i virtuální).

Po ukončení funkce `setup()` přejde program do funkce **`loop()`**, která běží v nekonečné smyčce. Toto dělení je specifické pro Arduino, respektive pro jím používaný framework *Wiring*, což je rozšíření jazyku C o použití s mikrokontroléry. Běžně se na MCU vyskytuje pouze jedna funkce `main()` která v sobě obsahuje jak prvotní nastavení, tak hlavní programovou smyčku, kterou si ale musí programátor vytvořit sám.

4.4.2 Softwarový restart mikrokontroléru

Má-li zařízení obsahovat restartovací tlačítko je nutné ho umět i obsloužit. Pokud by stačil pouhý restart MCU, je implementace jednoduchá – MCU je přímo vybaven pinem, který po přivedení napětí restartuje mikrokontrolér. Jenže v případě, že má mít tlačítko více funkcí v závislosti na délce stisku, je nutné to řešit jinak.

ATMega neobsahuje přímo instrukci pro restart. V jejím případě existují dvě cesty, jak toho dosáhnout. Buď lze využít watchdog nebo skok v paměti. Použití watchdogu je snadné – program se cíleně uvede do nekonečné smyčky a watchdog ho po chvíli sám restartuje (samozřejmě musí být před tím aktivován). Druhá možnost je jednodušší a rychlejší. Spočívá ve skoku na začátek programu, tedy na adresu 0. [26] Nejedná se sice přímo o restart MCU (například nejsou nulovány registry), ale je znovu spuštěna smyčka setup včetně nové inicializace paměti RAM, což je často dostačující řešení. Funkce samotná pak může vypadat například takto

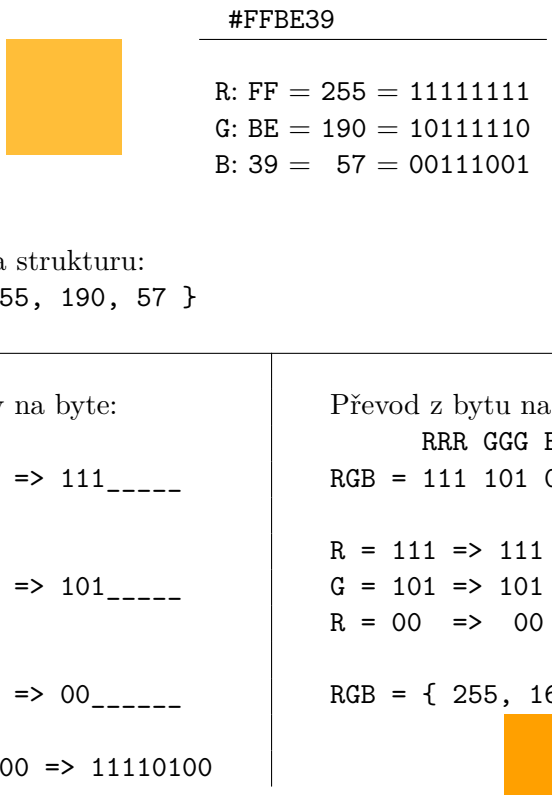
```
inline resetArduino() { asm volatile ( "jmp 0"); };
```

4.4.3 Zpracování barev pomocí MCU

V rámci programu může být barva kódována třemi různými způsoby. Ve všech třech případech se používá RGB model s rozdílnou bitovou hloubkou. Uživatelské rozhraní generuje barvy ve tvaru `#RRGGBB`. To znamená, že barva je popsána hexadecimálními čísly, přičemž se jedná o 24-bitovou barevnou hloubku. Tento tvar je pro strojové zpracování zcela nevhodný, proto je ihned přeložen do struktury `RGBColor`, která ukládá každou složku zvlášť jako jeden byte, takže se stále jedná o 24-bitovou hloubku (8 bitů na jednu barvu), ovšem oproti předchozímu tvaru postačí pro uložení 3 byty místo původních 7.

Reprezentace v podobě struktury je vhodná i z pohledu dalšího zpracování. PWM výstup Arduina má taktéž 8-bitovou hloubku, tudíž nejsou potřeba žádné další propočty a jednotlivé složky barvy lze rovnou zapsat na jednotlivé piny. Z důvodu optimalizace (viz 4.4.4) je barva dále komprimována na 8 bitovou hloubku. Zde už dochází ke ztrátě informací.

Uspořádání v rámci 8 bitů je `RRRGGGBB`. Pro (de)kódování se nepoužívá paleta, výsledné číslo je přímo vypočteno. Na obrázku 4.11 je názorně ukázán postup zpracování uživatelem zadané barvy až po komprimovaný 8-bitový formát a jeho převod zpět na 24 bitů. Podrobnější popis se nachází v příloze D.



Obrázek 4.11: Ukázka převodu barvy

4.4.4 Optimalizace ukládání dat

V případě ukládání většího množství dat je vhodné optimalizovat kolik místa v paměti zaberou. Jistě je jednodušší všechny čísla v kódu ukládat jako *integer* o velikosti 4 byty a nemuset řešit případné přetečení a chyby s ním spojené. Když se jedná o jedno či dvě čísla, ničemu to nevadí. Ale v případě, že je takových hodnot 5 v jedné struktuře a těch struktur v paměti existuje 20, tak už to v součtu zabere 400 bytů, což může být problém...

Proto je nutné při vytváření každé proměnné se pokusit odhadnout, jaký rozsah hodnot bude potřeba uchovávat a podle toho vybrat vhodný datový typ. Pro názornou ukázku jsem si vybral strukturu `Light`, která slouží k uchování informaci o jednom světle. Mezi informace, které je třeba uchovat patří

- **Název** slouží pro snadnější orientaci uživatele – *32 bytů*
- **Typ** konkrétně, zda je světlo stmívatelné, běžné nebo RGB – *1 byte*

- **Pin** na kterém je světlo připojeno. V případě RGB je třeba uchovat tři piny – *3 byty*
- **Aktuální stav** tedy *zapnuto* nebo *vypnuto* – *1 byte*
- **Index** v seznamu – *1 byte*
- **Aktuální jas a barva** – *1 + 3 byte*

To by znamenalo, že výsledná struktura by byla až 42 bytů veliká, což jsou 2 % paměti RAM. Při optimalizaci je vhodné začít u hodnot, jež nejsou nutné pro funkčnost programu. Například index v seznamu je zcela zbytečný údaj. Ten se používá při přístupu k samotné struktuře, jeho vynecháním proto nepřijdeme o žádnou funkčnost.

V podobné situaci se nachází název světla. Tedy ne, že by byl tento údaj zbytečný, ale není nijak důležitý pro chod programu. Tuto informaci potřebuje znát pouze uživatel a to jen ve chvíli, kdy chce získat nastavení zařízení. Proto postačí nechat název uložený pouze v EEPROM paměti, odkud ho lze kdykoli v případě načíst a odeslat uživateli.

Další optimalizace se týká způsobu uchování aktuálního jasu a barvy. Teoreticky by tyto hodnoty nebylo vůbec nutné uchovávat, ale jsou potřebné pro možnost světlo vypnout a po nějakém čase opět zapnout se stejným nastavením. Ovšem je zbytečné tyto informaci ukládat do 4 bytů. Nemá smysl uchovávat jas a barvu zároveň, jelikož jas lze promítnout přímo do samotné barvy. Tu také není nutné uchovávat v 24-bitové hloubce, pro domácí osvětlení bohatě postačí 8-bitová hloubka (stále se jedná o 265 různých kombinací barvy a jasu). Tím pádem lze do jednoho byte uložit buď aktuální jas (u jednobarevných světel) nebo barvu v případě RGB osvětlení.

Obrázek 4.12: Spojení jasu a barvy

Uchovávat pin jako 3 byty není zcela nutné. Je možné, aby i RGB osvětlení mělo definovaný pouze jeden pin. Čísla zbylých dvou pinů pak budou závislé na tom prvním. Vyřešil jsem to tak, že index se nastavuje pro červenou složku. Zelená složka má pin o jedna větší, modrá o dva. Tedy připojí-li se na pin č. 3 červená složka, zařízení automaticky zvolí pin 4 pro zelenou a pin 5 pro modrou.

Jelikož informace lze uchovávat nejen na úrovni bytů, ale i bitů, tak je možné optimalizovat i zbylé údaje. Např. aktuální stav je typu *boolean*, což znamená, že uchovává jen dvě hodnoty 0 a 1. Tudíž z bytu, který má k dispozici (MCU neumí pracovat s menší jednotkou) využívá pouze 1 bit ($\log_2 2 = 1$). Zbylých 7 bitů je nevyužitých a lze do nich zapsat jiné hodnoty. Podobně typ, který nabývá pouze tří hodnot ke svému popisu potřebuje 2 bity ($\log_2 3 \doteq 2$), respektive do dvou bitů lze uložit až 4 (2^2) různé informace, což jsem využil pro definici čtvrtého typu – *nezapojeno*. Stále zbývá 5 nevyužitých bitů. Ty nabízí prostor pro uchování až 32 (2^5) různých hodnot. Ty jsem využil pro uložení čísla pinu.

Pin					**Typ**		**S.**
7	6	5	4	3	2	1	0

Jas/barva							
7	6	5	4	3	2	1	0

Obrázek 4.13: Rozložení struktury `Light` v paměti

Ve výsledku struktura zabírá 2 byty místo původních 42, tedy pouhou 0,1 % paměti RAM. S ohledem na maximálně 32 připojených svítidel tak lze na začátku programu vy-

tvořit statické pole. To bude v paměti zabírat pouhých 64 bytů, což jsou nějaké 3 %. To je přiměřená cena za jednoduší program bez nutnosti dynamicky alokovat paměť a potenciální problémy s tím spojené.

Obrázek 4.14: Obsah struktury před a po optimalizaci

4.4.5 Optimalizace práce s řetězci

Další paměťově náročnou částí programu jsou textové řetězce. Před jejich použitím překladač automaticky zajistí načtení z FLASH do RAM, zejména z důvodu nižší rychlosti FLASH paměti. Toto chování je správné a ve většině případů jeden řetězec v paměti RAM nepřekáží. Jako problémové se ale ukázaly ladící výpisy na sériovou linku. Obsahovala-li jich funkce větší množství, tak se zaplnila paměť a program zkolaboval.

Řešení je jednoduché – přeskočit mezifázi s načítáním řetězce do RAM a rovnou ho číst z paměti FLASH. To lze provést pouze u funkcí, které řetězec nijak nezpracovávají, nýbrž pouze předávají dál (např. všechny „tisknoucí“ funkce). Pro usnadnění existuje `F()` makro. Na následujícím řádku je ukázka jeho použití:

```
Serial.print( F( "Hello word!" ) );
```

Podobně lze postupovat i při uchování větších řetězců v paměti. Například různých HTTP odpovědí. Takto uložená data je třeba označit klíčovým slovem `PROGMEM`, přičemž není nezbytně nutné se omezit pouze na řetězce, ale může se jednat třeba i o pole čísel (např. by bylo vhodné tímto způsobem uložit paletu barev). Ukázka uložení řetězce:

```
const char HELLO[] PROGMEM = "Hello word!";
```

Čtení takto uložených dat je mírně složitější, jelikož se nenachází v paměti RAM. K tomuto účelu slouží funkce `pgm_read_byte_near( address )`, která načte jeden byte z paměti FLASH, popř. její varianty pro jiné datové typy. Parametr *address* je součtem adresy začátku řetězce, který je uložen v ukazateli na řetězec a pořadím znaku. V následující ukázce znázorňuji přečtení prvních 3 znaků z řetězce.

```
char znak_0 = pgm_read_byte_near( HELLO     );
char znak_1 = pgm_read_byte_near( HELLO + 1 );
char znak_2 = pgm_read_byte_near( HELLO + 2 );
```

Použitím těchto metod lze dosáhnout výrazné úspory paměti RAM. V rámci testování ladicí výpisy zabírali až 50 % paměti.

4.5 Uživatelské rozhraní — umístění, struktura, funkčnost

Jak už bylo zmíněno v úvodu této kapitoly, uživatelské rozhraní je implementováno jako webová stránka. Při práci s ní musí uživatel znát IP adresu samotného zařízení. To je omezení dané jednoduchostí implementace. Technicky vzato by bylo možné doplnit systém o automatické vyhledávání, ale nepřišlo mi to jako klíčová vlastnost. V rámci testování byla do rozhraní přidána speciální adresa `0.0.0.0`, která ke své funkčnosti nepotřebuje fyzicky přítomné zařízení. Mimo to je nutné najít vhodné úložiště pro samotnou webovou stránku. V rámci práce byly testovány tři následující možnosti:

- **Paměť mikrokontroléru** se nabízí jako první variantou. Z pohledu uživatele je to ideální řešení – nemusí nic nastavovat, stahovat apod., pouze se připojí na zařízení. Z pohledu programátora je to naopak ta nejkomplikovanější varianta. Dnešní webové stránky mohou mít velikost i několik megabajtů, ale tolik paměti MCU nemá k dispozici. A i ta nejjednodušší stránka bez obrázků pouze s kaskádovými styly a jednoduchým JavaScriptem zabere několik kilobajtů. Což by už bylo reálné nahrát do paměti MCU, ale znamenalo by to se vzdát většiny současných technologií (jQuery, fonty s ikonami, složitější kaskádové styly, apod.).

  Alternativou je použití externí FLASH paměti, díky čemuž odpadá problém s nedostatečnou kapacitou. Jenže stále je tu druhá, vcelku významná, překážka v podobě nízké komunikační rychlosti samotného Wi-Fi modulu. Ten je schopný odesílat jeden znak za 20 ms. Při vyšší rychlosti dochází k zahlcení sériové linky. Snadným propočtem zjistíme, že jednoduchá stránka o velikosti 100 KiB by se odesílala cca. půl hodiny, což je nepoužitelné.

- **Externí server** řeší výše uvedené problémy s rychlostí přenosu a velikostí úložiště. Jenže konečnému uživateli to přinese pouze komplikace. Stává se závislý na zmíněném serveru a funkčnosti internetového připojení. Pokud jedno z toho selže, přijde o možnost ovládat osvětlení. Mimo to se nemůže bránit nechtěné aktualizaci uživatelského rozhraní která může z důvodu chyby znemožnit celé ovládání.

- **Lokální úložiště** je kompromisem předchozích dvou řešení. Sice nedosahuje výhod uchování v paměti MCU, jelikož webová stránka musí být uložena na všech zařízeních, které mají sloužit k ovládání. Na druhou stranu již není nezbytně nutné internetové připojení k ovládání osvětlení. Ideálním řešením je kombinace tohoto přístupu s předchozím bodem (externím serverem), kdy uživatel primárně využívá stránku uloženou na externím serveru. V případě výpadku má k dispozici plně funkční lokální kopii.

Já jsem si vybral kombinaci druhé a třetí možnosti. Umístění na externím serveru je jednodušší z pohledu vývoje, lze takto snadno testovat funkčnost skrze různá zařízení bez nutnosti přenosu souborů.

4.5.1 Struktura webové stránky

Webová stránka je složena ze dvou hlavních bloků. Část, kterou vnímá uživatel se nazývá *frontend*. Ta je tvořena strukturovaným souborem psaným v jazyce html, který definuje všechny prvky obsažené na stránce. Aby byl výsledek funkční i po grafické stránce je tento soubor doplněn o tzv. kaskádové styly. Naproti tomu je třeba webovou stránku oživit, aby reagovala na pokyny uživatele a byla schopná po síti komunikovat se zařízením. K tomu slouží *backend* naprogramovaný v jazyce JavaScript za využití jQuery knihovny.

Frontend

Na obrázku 4.15 je ukázka hotového grafického prostředí zobrazeného na mobilním zařízení, vykreslení na počítači je velmi podobné, viz obr. A.6. Každé světlo je zde reprezentováno jako samostatný funkční blok. Skrze něj je možné světlo zapnout/vypnout, změnit jas či barvu a provést nastavení (číslo pinu, typ, jméno). To vše je přizpůsobeno i pro ovládání dotykem (ovládací prvky jsou větší). Skrze tyto funkční prvky jsou zároveň uživateli sdělovány informace o aktuálním stavu.

V záhlaví stránky je pak vstup pro zadání IP adresy zařízení (automaticky se předvyplňuje poslední zadaná) s možností aktualizace stavu a tlačítko nastavení samotného řídícího zařízení, tedy spíše Wi-Fi modulu.

Obrázek 4.15: Ukázka uživatelského prostředí na mobilu

Backend

Dříve měl backend na starosti kompletní oživení webové stránky. Tedy například kontrolu hodnot ve formulářích. Tuto úlohu dnes již přebrala nová specifikace HTML5. Snadno lze definovat přesný tvar vstupu, čehož využívám např. při zadávání IP adresy. Nový standart CSS3 naopak přinesl možnost animace prvků. Snadno tak lze např. reagovat na pohyb myši, či pohybovat s prvky bez použití JavaScriptu, příkladem budiž animace zpracování příkazu.

Samotný JavaScriptový kód se stará o generování stránky, odesílání příkazů a dekódování odpovědí. To vše probíhá asynchronně technologií AJAX, tedy na pozadí je odeslána žádost a zpracována odpověď bez toho, aniž by byl uživatel přesměrován na jinou stránku nebo omezen v práci. Nejdříve je tedy nutné načíst nastavení a stav osvětlení, což je první akce, kterou webová stránka provede (tedy až poté, co od uživatele nebo z paměti získá IP adresu zařízení). Odpovědí jsou přímo binární data z paměti RAM (konkrétně se jedná o pole struktur `Light`).

Na základě těchto dat je vygenerován obsah samotné webové stránky. Tedy co jedna struktura `Light`, to jeden blok s osvětlením. Poté jsou na interakce uživatele (posun ukazatele jasu, kliknutí na ikonu žárovky, apod.) navázány jednotlivé funkce, které opět generují AJAX dotazy a na pozadí posílají příkazy do zařízení a reagují na jeho odpovědi.

Zde by bylo vhodné zmínit problémy, které nastanou při umístění webové stránky na lokální či externí úložiště. Bezpečnostní politika některých internetových prohlížečů (testováno v prohlížeči Google Chrome) zakazuje asynchronně načítat data z jiného zdroje, než je aktuální. To znamená, že v případě, kdy je webová stránka uložena v paměti PC, tedy na adrese `localhost`, tak není možné požádat o data na adrese `192.16.4.1`. Existuje jediný způsob, jak to provést. Respektive, existují dva, ale ten druhý způsob spočívá ve vypnutí těchto pravidel, což představuje bezpečnostní riziko.

Řešením je negenerovat žádost pomocí AJAX, ale za běhu vložit novou referenci na soubor do hlavičky html souboru, což způsobí, že jej prohlížeč stáhne a spustí. Aby tato funkce mohla nějakým způsobem předávat data, používají se tzv. *callback* funkce. Ty přijímají data jako parametr a nějakým způsobem je zpracují (příklad na obrázku 4.16). [21]

```
function zmenaJasu( odpoved ) {
        if ( odpoved.uspesnost == true ) {
                rozsvitIkonku();
        }
        else {
                vypisChybu();
        }
}
```

Obrázek 4.16: Příklad callback funkce

Tento parametr se nejčastěji ukládá ve tvaru JSON, což je vhodné pro budoucí zpracování. Občas se tento způsob tak nazývá jako JSONP (*JSON in Padding*). Příklad možného volání callback funkce, tedy obsah souboru, který se asynchronně načetl znázorňuje obrázek 4.17.

```
zmenaJasu( { uspesnost : true } );
```

Obrázek 4.17: Příklad JSONP odpovědi

Komunikace se zařízením proto musela být mírně přepracována. Místo HTML stavových kódů jsou odpovědi ve tvaru JSONP, přičemž z důvodu co nejrychlejší komunikace se používá standart HTML 0.9 – tedy pouze data bez hlavičky. Odpověď je také komprimována na co nejmenší možnou velikost, což je 11 bytů. Obsluha v kódu je pomocí callback funkcí. Aby se zamezilo zahlcení Wi-Fi modulu příkazy, tak není možné vygenerovat novou žádost, dokud není zpracována předchozí nebo nevypršel časový limit.

4.5.2 Jména pro jednotlivé osvětlení

Zatímco pro stroje je vhodnější pojmenovávat objekty čísly (indexy), tak lidé preferují jmenné názvy, které jsou snadněji zapamatovatelnější. Proto mělo původně každé světlo mít i své jméno. To mělo být uchováno v rámci paměti mikrokontroléru, aby bylo možné tyto data snadno synchronizovat skrze zařízení.

Jenže nízká komunikační rychlost Wi-Fi modulu tuto myšlenku poměrně zkomplikovala. Proto jsem se uchýlil ke kompromisu, kdy je jméno uchováno pouze v paměti uživatelského rozhraní, tudíž jej není nutné vůbec přenášet skrze síť.

4.6 Rozšířený návrh zapojení

Zapojení uvedené v podkapitole 4.1 je určeno pouze pro testování a vývoj. Po jeho ukončení bylo provedeno několik rozšíření. Indikační diody jsou nahrazeny svorkami pro připojení libovolného osvětlení. Zároveň je rozšířen počet výstupních pinů na 10 a jsou posíleny pomocí NPN zesilovačů. Zařízení by mělo umožňovat jednoduché interakce s uživatelem i během nefunkčního Wi-Fi spojení, proto je přidáno tlačítko pro restart a informační diody indikující aktuální stav. Takto rozšířené zapojení stále situované v nepájivém poli se nachází na obrázku 4.18 (schéma je v příloze A.3).

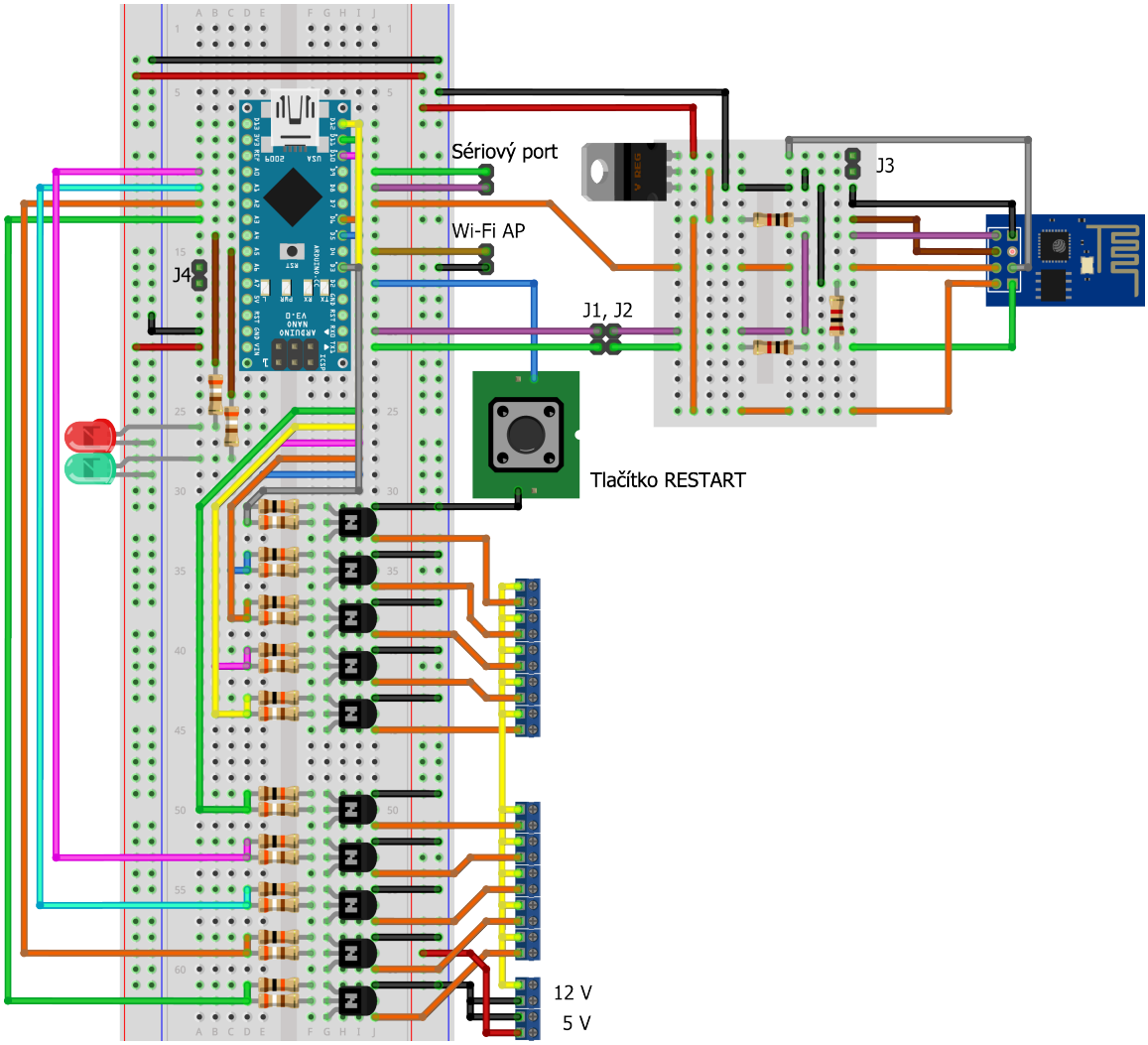
4.6.1 Doplnění o NPN tranzistory

Posílit výstup pinů tranzistorem není nijak složité. Přesto je třeba vybrat vhodný tranzistor a spočítat hodnotu bázového rezistoru. Doporučená zátěž jednoho pinu je 20 mA, přičemž zátěž všech pinů v součtu nesmí překročit 200 mA. Aby nebyl MCU zbytečně zatěžován, určil jsem maximální proud na jeden pin 15 mA. Při deseti pinech co jsou k dispozici to znamená 150 mA, zbylých 50 mA slouží jako rezerva a pro napájení indikačních LED.

Dále je nutné definovat spínaný proud. Pro většinu osvětlení by mělo stačit 0,5 A. Na základě těchto hodnot již lze pokročit k výběru tranzistoru. Ten musí být konstruován minimálně na 0,5 A (raději o něco více), přičemž jeho proudový zesilovací činitel musí být minimálně 100 podle rovnice pro bázový proud (I_B)

$$\begin{aligned} I_B &= \frac{I_C}{h_{FE}} \cdot 3 \\ h_{FE} &= \frac{I_C}{I_B} \cdot 3 \\ h_{FE} &= \frac{0,5}{0,015} \cdot 3 = 100 \end{aligned}$$

Těmto podmínkám vyhovuje např. tranzistor BC337-40. Hodnota bázového rezistoru je určena na základě informací z datasheetu [14] a následujícího vzorce, přičemž záleží,



Obrázek 4.18: Rozšířené zapojení

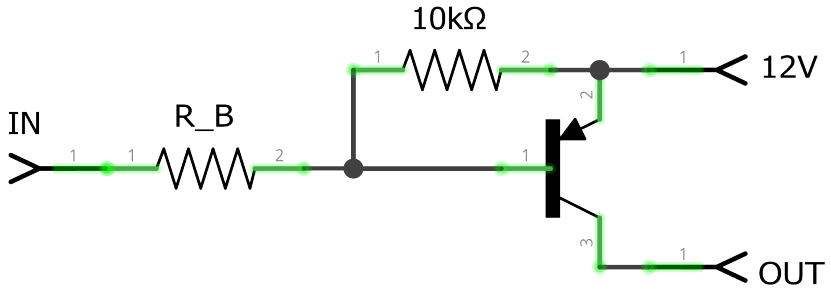
jestli je MCU napájen 5 nebo 3,3 volty. Níže jsou uvedeny výsledky pro obě napětí včetně nejbližšího běžně dostupného rezistoru.

$$
\begin{aligned}
R_B &= \frac{V_{IN} - V_{BE}}{\frac{I_C}{h_{FE}} \cdot 3} \\
R_B &= \frac{5 - 1,2}{\frac{0,5}{150} \cdot 3} = 380\,\Omega \doteq 390\,\Omega \\
R_B &= \frac{3,3 - 1,2}{\frac{0,5}{150} \cdot 3} = 210\,\Omega \doteq 220\,\Omega
\end{aligned}
$$

Uzemněním báze pomocí rezistoru (s hodnotou 10 kΩ) zabraňuje slabému probliknutí při startu, které by mohlo způsobovat spouštění mikrokontroléru, jelikož v tu chvíli není pin nikam připojen. [12]

NPN tranzistory mohou představovat určitý problém při práci s RGB diodami. Výše uvedené zapojení je vhodné pro uspořádání se společnou anodou (1 pin pro kladné napětí,

3 piny pro zem), naopak uspořádání se společnou katodou takto ovládat nelze. Aby to bylo možné, je nutné na výstupní pin doplnit PNP tranzistor tak, jako na schématu 4.19. [12]



Obrázek 4.19: PNP spínač pro zapojení se společnou katodou

4.6.2 Napájení

Požadovat po uživateli, aby vybavil zařízení třemi různými zdroji (3,3 V pro Wi-Fi modul, 5 V pro Arduino a 12 V pro LED osvětlení) nelze. Proto je třeba je nějakým způsobem zredukovat. Jak už jsem nastínil v úvodu, zbavit se napětí 5 V není až takový problém, naopak nám to vše mírně zjednoduší. Je nutné na desce Arduina přemostit 5V napěťový filtr a naopak celý obvod doplnit o 3,3V filtr. Pak by bylo možné připojit Wi-Fi modul na sériový port Arduina přímo bez napěťového děliče.

V zapojení 4.10 tato změna není provedena. Zejména proto, aby bylo možné zařízení sestavit z běžně dostupných součástek bez potřeby dalších úprav. Napájení pro Wi-Fi modul již není řešeno externě, ale za pomoci napěťového filtru. Stále zde tedy figurují dvě různé napětí, 5 V pro řídící zařízení a 12 V pro osvětlení. Tuto vlastnost by bylo v případě komerční výroby nutné změnit.

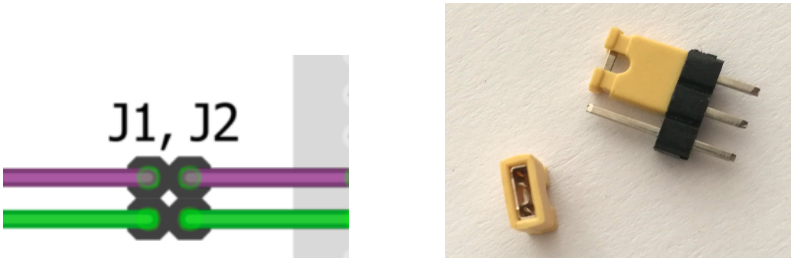
4.6.3 Možnost nahrávání nového firmware

Jelikož se stále jedná o prototyp, měla by být zachována možnost nahrát nový program. Aby to bylo možné, musí být umožněn přístup k sériovému portu. A to jak na Arduinu, tak na Wi-Fi modulu. K tomu slouží piny `J1` a `J2`. Pro správnou funkcionalitu musí být přemostěny pomocí *zkratovací propojky* (též nazývané *jumper*). Naopak v případě, že bude potřeba do Arduina nahrát nový program, je nutné spojení přerušit, čímž se zabrání tomu, aby Wi-Fi modul vstupoval do komunikace. Pro propojení s PC lze buď využít USB port na Arduinu, nebo použít přímo na piny `RX` a `TX` vyvedené na piny `J1` a `J2`.

Obdobně lze nahrát nový firmware do Wi-Fi modulu. Zde je navíc potřeba uzemnit pin `GPIO_0` pomocí propojky `J3`. Za zmínku ještě stojí piny `J4`, které v tuto chvíli nemají žádný význam. Přidal jsem je tam pouze s ohledem na možný budoucí vývoj, jelikož se jedná o vstupy A/D převodníku.

4.6.4 Interakce s uživatelem

Potřebu nastavit zařízení i bez Wi-Fi připojení pocítí uživatel zejména ve chvíli, kdy toto spojení selže. Typicky se může jednat buď o chybné nastavení nebo výpadek. Měla by tudíž existovat možnost, jak provést restart programu popř. i jeho nastavení. Proto je většina

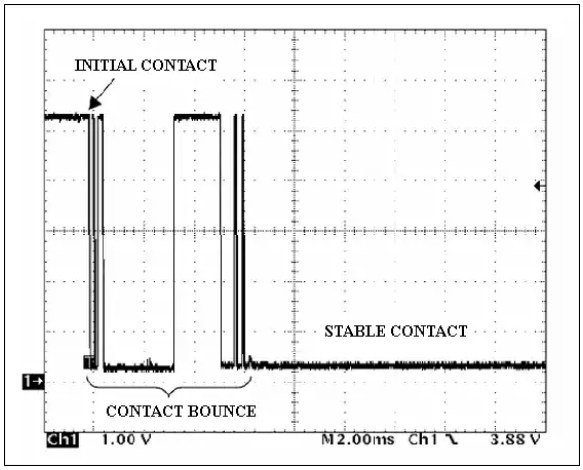


(a) Piny `J1` a `J2` (b) Zkratovací propojky

Obrázek 4.20: Přístup k sériovým portům

podobných zařízení vybavena restartovacím tlačítkem. Aby toto tlačítko fungovalo za všech situací, musí být obsluhováno pomocí přerušení.

Mimo restartu může tlačítko vykonávat více činností a to na základě délky stisku. Jako užitečné my přišli následující. Krátký stisk (více jak 500 ms) provede pouhý restart zařízení. To je vhodné, pokud by se program nějakým způsobem zacyklil. Podmínka na 500 ms je tam ze dvou důvodů. Normálně rychlý stisk tlačítka trvá cca 200 ms, to znamená, že je poměrně malá šance, aby uživatel stiskl tlačítko omylem. Druhým důvodem jsou tzv. *zákmity* či *záchvěvy*, které vznikají při každém stisknutí tlačítka. Tlačítko je totiž elektromechanický prvek, a jako takový nefunguje zcela ideálně a i při krátkém stisknutí vygeneruje několik signálů (viz obr. 4.21). Ty je nutné nějakým způsobem filtrovat a to buď pomocí speciálního obvodu nebo právě časovačem.



Obrázek 4.21: Zákmity vzniklé při stisknutí tlačítka [1]

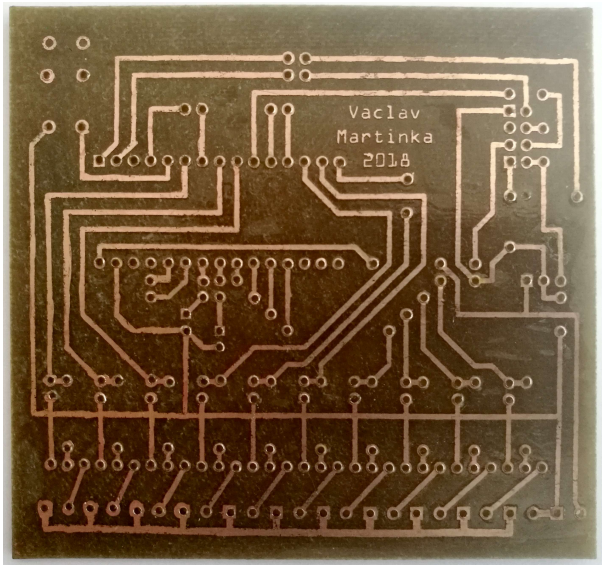
Delší stisk (aspoň 5 s) vyvolá restart nastavení Wi-Fi sítě, ale zůstane zachováno nastavení osvětlení. A nakonec stisk delší jak 10 s restartuje veškeré uživatelem uložené nastavení. Mimo to jsem dále doplnil piny *Wi-Fi AP*. Jsou-li tyto piny během startu propojeny, tak se zařízení dočasně přepne do režimu vysílání vlastní Wi-Fi sítě a to až do příštího restartu. Toto chování je výhodné v případě dočasného výpadku sítě.

4.6.5 Návrh DPS a seznam součástek

Na závěr vývoje bylo zařízení přeneseno z nepájivého pole na DPS (desku plošných spojů). Její podklady pro návrh se nachází v příloze na obrázcích A.4 a A.5 nebo ve složce `DPS` na přiloženém CD. Jedná se o jednostrannou desku, která by měla být snadno vyrobitelná i v domácích podmínkách, což dokazuje obrázek 4.22 znázorňující doma vyrobenou DPS. Konečný seznam všech součástek obsahuje tabulka 4.2.

Rozšířený návrh		
MCU	Arduino Nano	1×
Wi-Fi	ESP8266-01	1×
Napěťový filtr	3,3 V	1×
Rezistor	390 Ω	12×
	1 kΩ	2×
	2 kΩ	1×
	10 kΩ	10×
Tranzistor	BC337-40	10×
LED	Červená	1×
	Zelená	1×
Pin	Kolík	12×
Svorkovnice	Po dvou pinech	12×
Tlačítko		1×

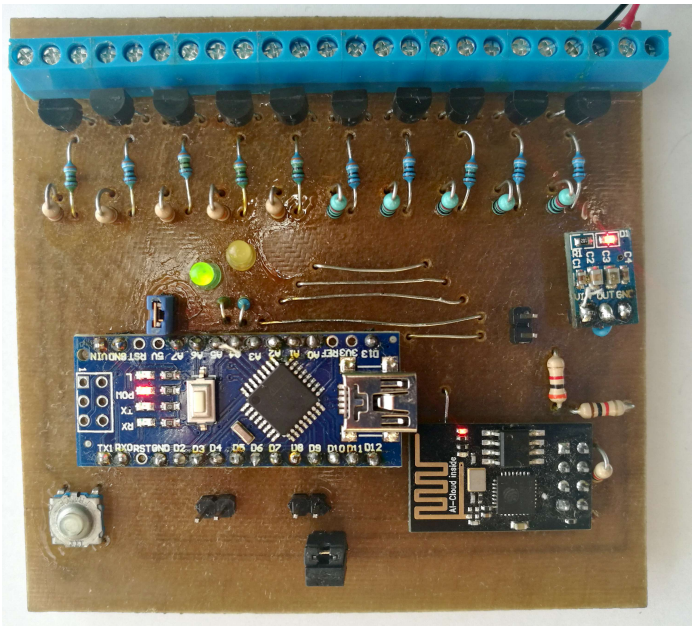
Tabulka 4.2: Seznam potřebných součástek



Obrázek 4.22: Deska plošných spojů

4.6.6 Spotřeba

Spotřeba je důležitý aspekt zařízení. Zejména jedná-li se o zařízení, které je neustále v provozu. Naměřená spotřeba prototypu v klidovém stavu je cca 80 mA. To díky tomu, že je MCU uspán a běží pouze Wi-Fi modul. Spotřeba v běžném provozu může vystoupat i přes 250 mA v závislosti na počtu připojených světel, jejich odběru a aktivitě Wi-Fi modulu.



Obrázek 4.23: Hotový prototyp

4.7 Testování a bezpečnost

Na závěr práce uvedu informace o testování zařízení, tedy co vše by mělo být testováno a co jsem skutečně testoval. S tím lehce souvisí zabezpečení zařízení, které by také mělo být otestováno, jelikož u každého zařízení schopného komunikace s okolním světem je třeba brát v potaz i možný útok, který je o to pravděpodobnější, je-li zařízení připojené do Internetu.

4.7.1 Možné scénáře útoku

Prošel jsem několik možných scénářů napadení, zjistil potenciální rozsah škod a navrhl řešení, jak takovému útoku předejít.

Prolomení přístupu do sítě

Tento typ útoku je možný pouze v případě, kdy Wi-Fi modul běží v režimu přístupového bodu. To znamená, že zařízení vytváří a spravuje celou Wi-Fi síť. Tento režim je určen především jako dočasný pro prvotní nastavení, ale samozřejmě je možné v něm plnohodnotně ovládat osvětlení. Toho může zneužít potenciální útočník, jehož cílem nebude ani tak ovládnout osvětlení, jako získat přístup do sítě.

Útočník buď využije slabinu v podobě slabého či dokonce žádného zabezpečení sítě. Popř. by teoreticky mohla být chyba v použitém firmware, což by v kombinaci s nevýkoným

hardware bez ochran před napadením (jako je např. firmware) mohlo vést k prolomení jakéhokoli zabezpečení. Je třeba zdůraznit, že se bavíme o síti vytvořené malým modulem ESP8266 stojícím ani ne dva dolary, u kterého nikdo nepočítá s tím, že by sloužil k vytvoření domácí Wi-Fi sítě.

Pokud by byl útok úspěšný, útočník získá nejen plný přístup k osvětlení (tomuto problému se věnuji dále), ale i do celé sítě. Může odposlouchávat veškeru komunikaci a případně se pokusit napadnout i další zařízení. Navíc takový útočník je v síti v podstatě neviditelný a uživatel ani nemusí tušit, že byla jeho síť napadena.

Prevence před útokem tohoto typu je jednoduchá – neprovozovat zařízení v režimu přístupového bodu. Dále by bylo možné zakázat vytváření sítě bez zabezpečení či zavést podmínku na minimální sílu hesla. Jistě by bylo vhodné také doplnit uživatelské rozhraní o informaci a počtu připojených zařízení, jejich IP adresách a času, kdy se naposled přihlásili, popř. vytvářet historii přístupů.

Získání přístupu k zařízení

V tuto chvíli je zařízení navrženo tak, že spoléhá na zabezpečení sítě a samo již žádné zabezpečení neimplementuje. To znamená, že pokud útočník získá přístup do sítě, zároveň získá plnou kontrolu nad tímto zařízením. Toho lze využít různě. V lepším případě bude útočník pouze škodit. Změní nastavení (či ho rovnou smaže), může rozsvěcet světla jak se mu zlíbí apod. V případě, že je uživatel přítomen doma, snadno to zjistí a může zakročit. Horší scénář nastává v situaci, kdy se útočník rozhodne skrývat a osvětlení využije k špionáži. Může tak vysledovat návyky členů v domácnosti a toho využít pro vydírání nebo jako tip pro zloděje. A to bez nutnosti být fyzicky na místě, jediné co je potřeba je přístup do sítě.

Navržené řešení je snadné – podmínit přístup do zařízení heslem. To by ovšem znamenalo vytvořit systém rolí s různými právy a doplnit příkaz pro přihlášení a další operace s tím spojené (přidávání uživatelů, změna oprávnění, apod.). Navíc by bylo nutné provozovat šifrovanou komunikaci mezi uživatelem a zařízením, aby nebylo možné odposlechnout heslo nebo ho podvrhnout. To vše navíc pouze přidá útočníkovy do cesty další překážku, ale nezajistí to naprostou bezpečnost. Mimo to by bylo opět vhodné vytvářet historii přístupů jako v předchozím bodě.

Odposlech hesla k domácí síti

Jedná se o velmi specifickou slabinu v zabezpečení. V případě, kdy uživatel mění nastavení Wi-Fi modulu, tak skrze síť posílá její SSID a heslo v nešifrované podobě. Pokud tato komunikace navíc probíhá po nezabezpečené síti, ke které se může každý připojit, je velmi jednoduché toto heslo odposlechnout a zneužít. Řešit to lze buď šifrováním nebo zákazem posílat heslo po nezabezpečené síti (popř. obojím).

4.7.2 Testování

Testování je důležitou součástí vývoje softwaru. Jen tak mohou být odhaleny chyby dříve, než napáchají nějaké škody. Testování softwaru je široké a komplexní téma, které by vydalo na samotnou práci. V následujících řádcích proto přiblížím, jak jsem testoval tuto práci.

Nejdříve je nutné si ujasnit, co vše je třeba otestovat. V rámci práce bylo vytvořeno několik relativně samostatných bloků. Každý z nich byl testován během jeho vývoje a následně i ve spojení s ostatními prvky. Dále je nutné vzít v potaz fakt, že testováním nelze prokázat bezchybnost programu, ale pouze to, že chyby obsahuje.

Program pro MCU je hlavní a zároveň nejhůře testovatelnou částí práce. Jeho funkčnost byla ověřována pomocí ladících výpisů na sériovou linku PC. S ohledem na omezené ladící prostředky MCU a náročnost tvorby případných automatických testů jsem testy prováděl ručně pro každou nově vytvořenou funkci. Správnost výsledků jsem ověřoval porovnáním hodnot z výpisu s očekávanými, odposlechem komunikace MCU s Wi-Fi modulem a měřením napěťových úrovní na výstupních pinech.

Uživatelské rozhraní nabízí bohatší možnosti testování. Hlavní výhodou je nezávislost na fyzickém zařízení. Naopak simulací komunikace lze snadno vytvořit (a otestovat) libovolnou situaci. Webové prostředí bylo doplněno o speciální IP adresu `0.0.0.0`, která slouží právě pro testování.

Komunikace mezi MCU a uživatelem je sice jednoduchá, ale i tak je nutné ověřit, že je správně navržena. Tedy zda lze pomocí zvolené technologie (HTTP) přenášet data v tomto formátu. Dále je nutné si ověřit, zda navržené příkazy jsou dostatečné nebo nějaký chybí, popř. je nadbytečný.

Zapojení bylo nejdříve navrženo na PC a bylo tak nutné ověřit jeho funkčnost. Buď pomocí simulace nebo realizací. Já jsem se vydal druhou cestou – vše jsem zapojil na nepájivém poli a stejně tak jsem testoval i případná rozšíření.

Návrh DPS úzce souvisí s předchozím bodem. Dnes se DPS kreslí na PC (popř. je počítač generuje sám) a tak už při návrhu probíhá elementární ověření funkčnosti. Přesto je tu k testování hned několik aspektů. Kromě správnosti propojení součástek je to například jejich rozmístění. Tedy to zda se navzájem nepřekrývají, dostupnost konektorů popř. i to, zda nehrozí přehřívání.

Označit testování za úspěšné či neúspěšné v podstatě nelze. Pouze mohu konstatovat, že díky testování bylo odhaleno množství chyb s různou důležitostí a to jak v kódu pro MCU tak v uživatelském rozhraní. Všechny tyto chyby byly následně opraveny, tudíž lze prohlásit, že v tomto případě bylo testování přínosné. Chyby se nevyhnuly ani návrhu DPS. Ten obsahoval hned dvě, naštěstí obě vcelku snadno řešitelné bez nutnosti vyrábět novou desku.

Kapitola 5

Závěr

Cílem práce bylo vyvinout zařízení pro ovládání osvětlení s kterým by uživatel komunikoval skrze Wi-Fi síť. Během vývoje vznikl zcela funkční prototyp včetně uživatelského prostředí. Cíl práce byl tedy splněn.

Jako první bylo třeba nastudovat možné způsoby ovládání osvětlení a existující řešení. To jsem provedl a na základě takto získaných vědomostí byli navrženy vlastnosti výsledného zařízení, jeho základní zapojení a způsob bezdrátové komunikace. S uživatelem komunikuje pomocí jednoduchých HTTP příkazů skrze Wi-Fi síť. Mezi základní vlastnosti patří schopnost zapnout či vypnout světlo a možnost změnit jeho jas a barvu.

Nyní bylo možné přistoupit k vývoji prvního prototypu. Byla popsána základní myšlenka programu a mohli být programovány i jednotlivé funkce. Souběžně s tím probíhal návrh a implementace uživatelského prostředí. Během vývoje byli postupně odhaleny a následně i opraveny nedostatky prvotního návrhu (například komunikace mezi uživatelem a zařízením) a přidány některé vlastnosti rozšiřující výslednou funkčnost.

Prototyp využívá platformu Arduino Nano společně s Wi-Fi modulem ESP8266-01. Lze k němu připojit až 10 různých externích zdrojů světla, přičemž toto číslo je dáno pouze omezeným počtem pinů Arduina a lze ho bez zásahu do zdrojových kódů zvýšit až na 32 pouhou změnou použitého mikrokontroléru. Pokročilý uživatelé jistě ocení detailní popis komunikace, naopak pro běžného uživatele bylo vyvinuto ovládací prostředí ve formě webové stránky, coby co možná nejvíce multiplatformní řešení. Zařízení se připojuje buď do existující Wi-Fi sítě nebo si vytvoří vlastní. Lze se na něj připojovat z více zařízení a to i současně – v jednu chvíli lze obsloužit až 4 různá zařízení. Na závěr byla navržena a osazena deska plošných spojů.

Během práce jsem si ucelil znalosti o fungování sítí a vývoji webových stránek, ale především jsem získal spoustu zkušeností ohledně programování mikrokontrolérů a obecně práci s elektronikou. Myslím si, že bezdrátové ovládání osvětlení si brzy najde cestu do většiny domácností a jsem rád, že jsem měl možnost do těchto vod přispět i svým dílem.

V práci bych rád pokračoval. Zajímavé by mohlo být do konceptu zařadit i bezdrátové vypínače, kdy by bylo možné ovládat osvětlení bez nutnosti použít smartphone či počítač, ale postačil by klasický nástěnný vypínač. Z celého konceptu bych také rád vynechal Arduino a využil pouze Wi-Fi modul, respektive více modulů, jenže pak by bylo nutné nějakým způsobem elegantně vyřešit vyšší množství IP adres jednotlivých zařízení, přičemž ty se mohou v rámci domácí sítě dynamicky měnit, tudíž to není zcela triviální úprava.

Literatura

[1] Software Methods to Achieve Robust 1-Wire® Communication in iButton® Applications. Maxim Integrated Products, Inc., Září 2008, Čerpáno 30. 4. 2018.
URL https://www.maximintegrated.com/en/app-notes/index.mvp/id/159

[2] Světelná účinnost zdrojů světla. Vega s. r. o., Leden 2008, Čerpáno 2. 3. 2018.
URL http://www.stavebnictvi3000.cz/clanky/zarovka-usporna-zarovka-mnozstvi-svetla/

[3] *FT232R USB UART IC Datasheet, version 2.13*. FTDI Chip, 2015.

[4] Úvod do počítačové grafiky - Základy optiky, barevné modely. ITnetwork.cz, Březen 2015, Čerpáno 2. 3. 2018.
URL https://www.itnetwork.cz/grafika/uvod-do-pocitacove-grafiky-optika-modely

[5] Adresa URL může být v aplikaci Internet Explorer dlouhá maximálně 2 083 znaků. Microsoft, Duben 2018, Čerpáno 27. 4. 2018.
URL https://support.microsoft.com/cs-cz/help/208427/maximum-url-length-is-2-083-characters-in-internet-explorer

[6] *ESP8266 AT Instruction Set*. Espressif Systems, Únor 2018.

[7] Asleson, R.: *Ajax : vytváříme vysoce interaktivní webové aplikace*. Computer Press, Brno 2006, ISBN 80-251-1285-3.

[8] Author: IEEE Standard and American National Standard for Electrical and Electronics Diagrams. *IEEE 315-1971 (ANSI Y32.2-1970)*, Leden 1971.

[9] Berners-Lee, T.: The Original HTTP as defined in 1991. w3.org, Čerpáno 5. 4. 2018.
URL https://www.w3.org/Protocols/HTTP/AsImplemented.html

[10] Berners-Lee, T.; Fielding, R. T.; Masinter, L.: Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, RFC Editor, Leden 2005.
URL http://www.rfc-editor.org/rfc/rfc3986.txt

[11] Berners-Lee, T.; Fielding, R. T.; Nielsen, H. F.: Hypertext Transfer Protocol – HTTP/1.0. RFC 1945, RFC Editor, Květen 1996.
URL http://www.rfc-editor.org/rfc/rfc1945.txt

[12] Bezstarosti, J.: Tranzistor polopatě. RoboDoupě, Prosinec 2011, Čerpáno 7. 3. 2018.
URL http://robodoupe.cz/wp-content/uploads/2012/01/tranzistor_polopate.pdf

[13] Bidlo, M.: IMP: Principy sériové komunikace, sériová komunikační rozhraní, 2017, FIT VUT v Brně.

[14] Diotec Semiconductor: *BC337-xBK BC338-xBK datasheet*. 2010.

[15] Doleček, J.: *Moderní učebnice elektroniky - 1. díl*. BEN - technická literatura, Praha 2005, ISBN 80-7300-146-2.

[16] Doleček, J.: *Moderní učebnice elektroniky - 2. díl*. BEN - technická literatura, Praha 2005, ISBN 80-7300-161-6.

[17] Doleček, J.: *Moderní učebnice elektroniky - 3. díl*. BEN - technická literatura, Praha 2005, ISBN 80-7300-184-5.

[18] Elizabeth Castro, B. H.: *HTML5 a CSS3 : názorný průvodce tvorbou WWW stránek*. Computer Press, Brno 2012, ISBN 978-80-251-3733-8.

[19] Fielding, R. T.; Gettys, J.; Mogul, J. C.; aj.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, RFC Editor, Červen 1999.
URL http://www.rfc-editor.org/rfc/rfc2616.txt

[20] Filip, A.: WiFi modul ESP8266-01. Říjen 2017, Čerpáno 15. 12. 2017.
URL http://xanadu.khnet.info/esp8266.php

[21] Hořčica, A.: Co je to JSONP? Adam's Notepad^2, 2007, Čerpáno 2. 5. 2018.
URL http://notepad.jslab.net/clanky/co-je-to-jsonp.html

[22] Janík, D.: *Zdroje světla*. bakalářská práce, Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřící techniky, 2014.

[23] Josefsson, S.: The Base16, Base32, and Base64 Data Encodings. RFC 4648, RFC Editor, Říjen 2006.
URL http://www.rfc-editor.org/rfc/rfc4648.txt

[24] Köhre, T.: *Stavíme si bezdrátovou síť Wi-Fi*. Computer Press, Brno 2004, ISBN 80-251-0391-9.

[25] Kristol, D.; Montulli, L.: HTTP State Management Mechanism. RFC 2965, RFC Editor, Říjen 2000.
URL https://www.rfc-editor.org/rfc/rfc2965.txt

[26] lalle_calle: Reset arduino with code. Arduino Forum, 2010, Čerpáno 14. 3. 2018.
URL https://forum.arduino.cc/index.php?topic=49581.msg354263#msg354263

[27] Lampa, P.: Protokol HTTP 1.1. FIT VUT v Brně, Září 2002, Čerpáno 4. 4. 2018.
URL http://www.fit.vutbr.cz/~lampa/WWW/http11.html.cs

[28] Malý, M.: *HRADLA, VOLTY, JEDNOČIPY – Úvod do bastlení*. CZ.NIC, z. s. p. o., Praha 2017, ISBN 978-80-88168-26-3.

[29] Matoušek, D.: *Práce s mikrokontroléry ATMEL AVR ATmega16*. 4. díl., BEN - technická literatura, Praha 2006, ISBN 80-7300-174-8.

[30] Matoušek, P.: *Síťové služby a jejich architektura*. Publishing house of Brno University of Technology VUTIUM, 2014, ISBN 978-80-214-3766-1, 396 s.

[31] Maurerová, L.: *Vliv protokolu HTTP 2.0 na moderní datové sítě*. bakalářská práce, Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2014.

[32] Navrátil, P.: *Počítačová grafika a multimédia*. Computer Media, 2009, ISBN 80-86686-77-9.

[33] Rampelt, J.: PWM programming activity. Sirius microSystems, Čerpáno 28. 4. 2018. URL http://www.siriusmicro.com/chrp3/pwm-c.html

[34] Růžička, R.: IMP: Analogové vstupy a výstupy, 2017, FIT VUT v Brně.

[35] Růžička, R.: IMP: Generování hodinového signálu, modul MCG, spotřeba MCU, 2017, FIT VUT v Brně.

[36] Růžička, R.: IMP: Porty MCU, jednoduchý vstup a výstup, 2017, FIT VUT v Brně.

[37] Růžička, R.: IMP: Čítače a časovače, 2017, FIT VUT v Brně.

[38] Růžička, R.: IMP: Vestavné systémy – Úvod, 2017, FIT VUT v Brně.

[39] Růžička, R.: IMP: Vývojové prostředky, 2017, FIT VUT v Brně.

[40] W3Techs: Usage of HTTP/2 for websites. Květen 2018, Čerpáno 5. 4. 2018. URL https://w3techs.com/technologies/details/ce-http2/all/all

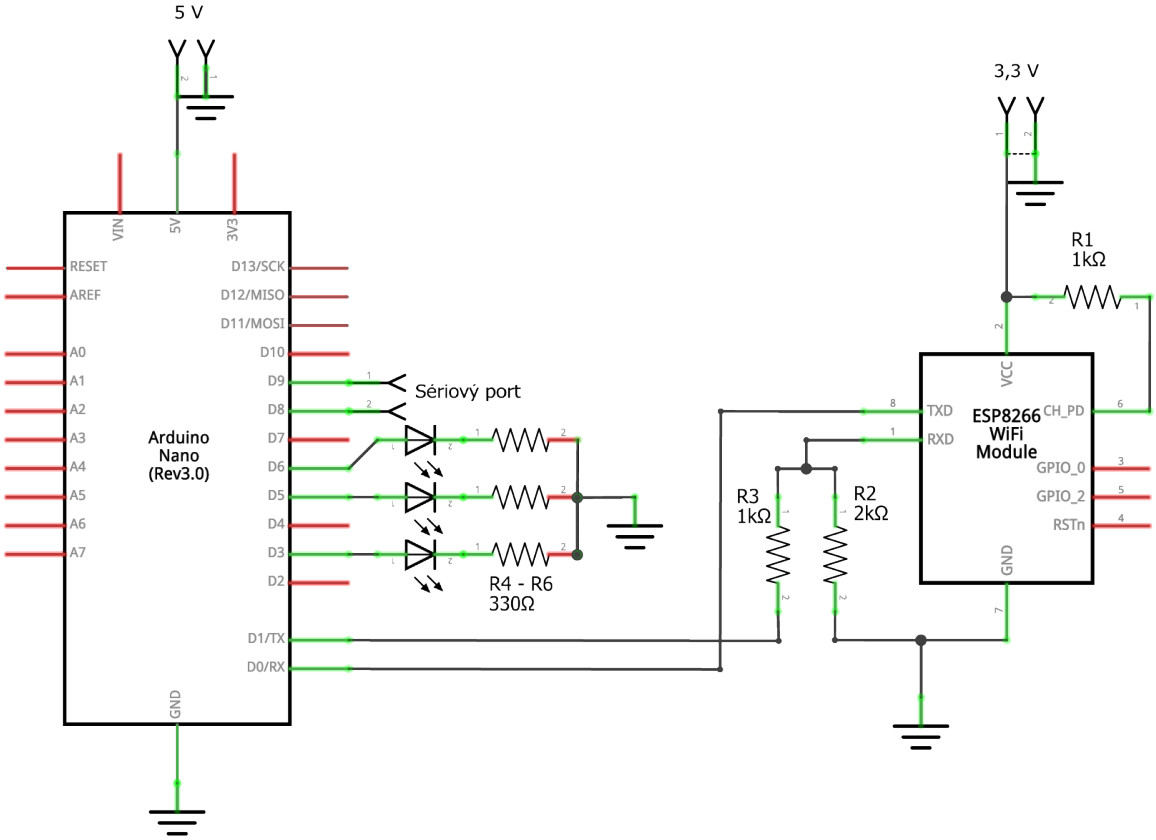
[41] Záviška, P.: *Metody pro zvýšení bitové houbky fotografií*. bakalářská práce, Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2015.

Přílohy

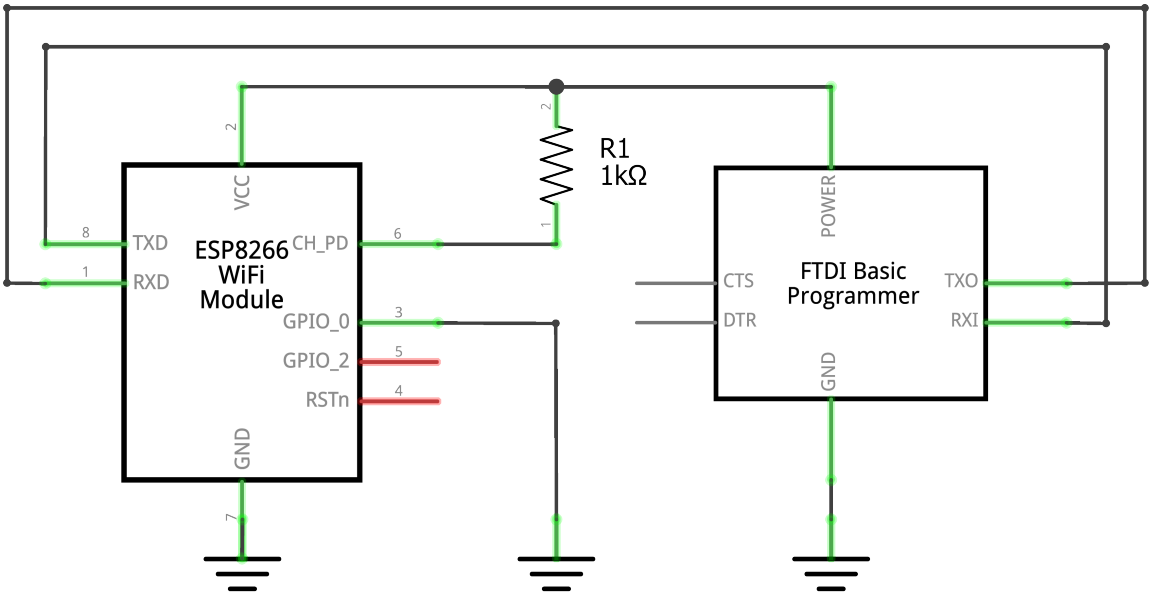
Seznam příloh

Příloha A

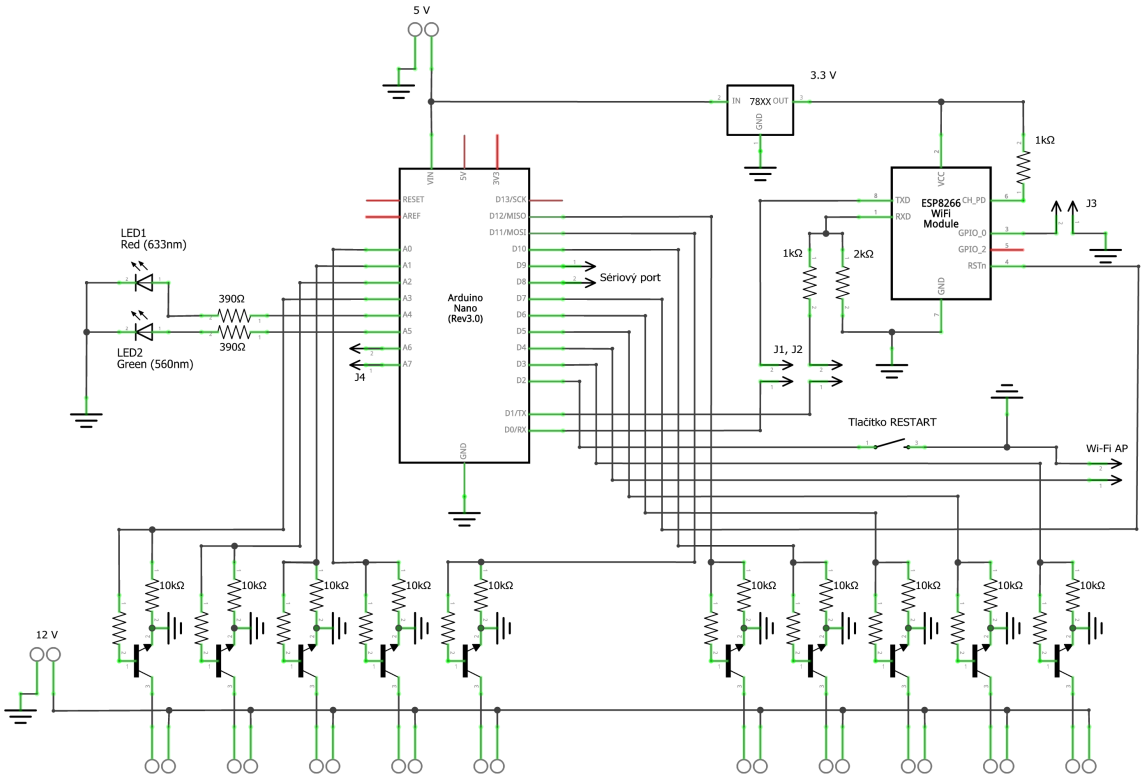
Schémata a obrázky



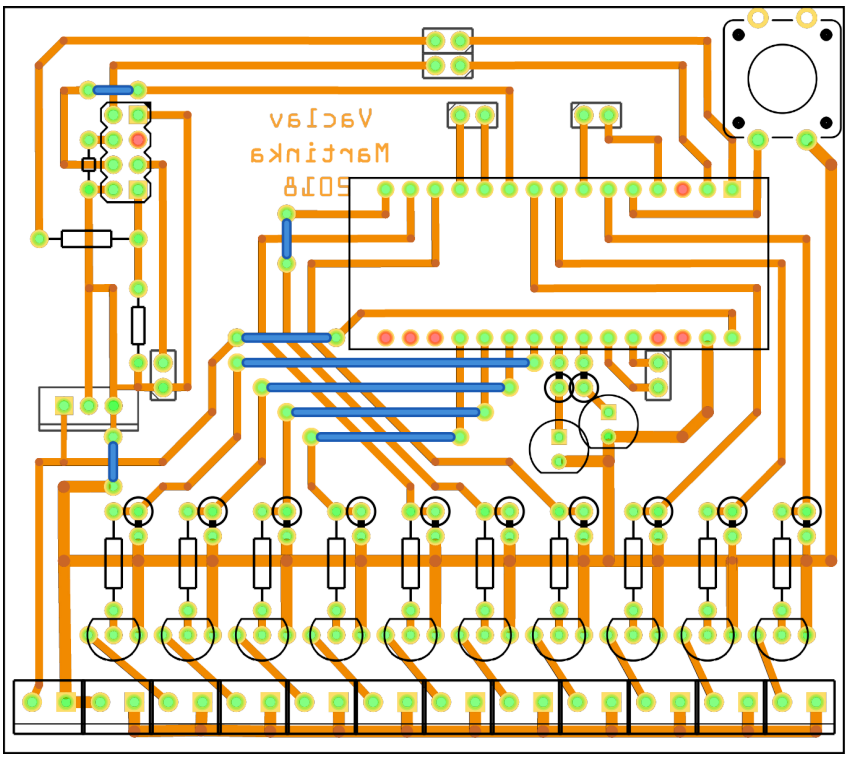
Obrázek A.1: Základní zapojení



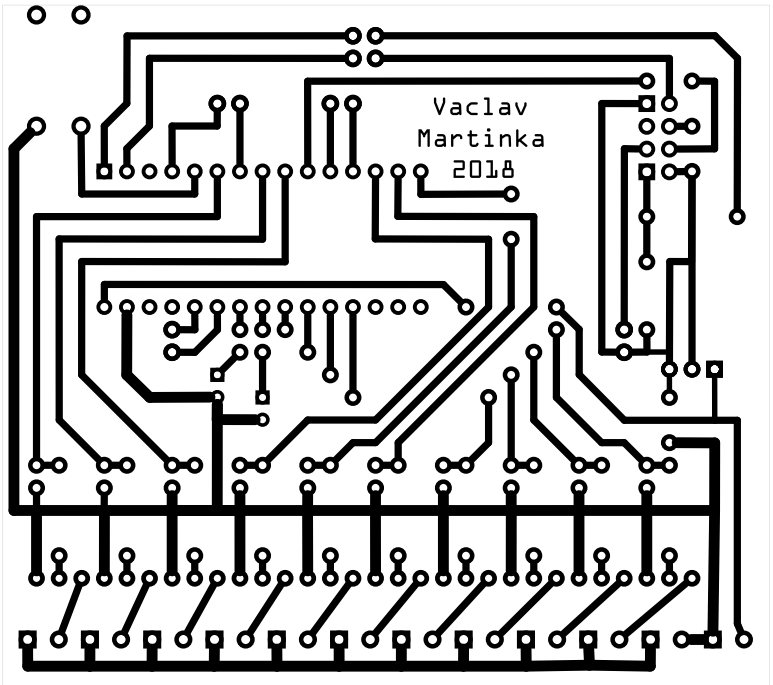
Obrázek A.2: Zapojení pro nahrání firmware



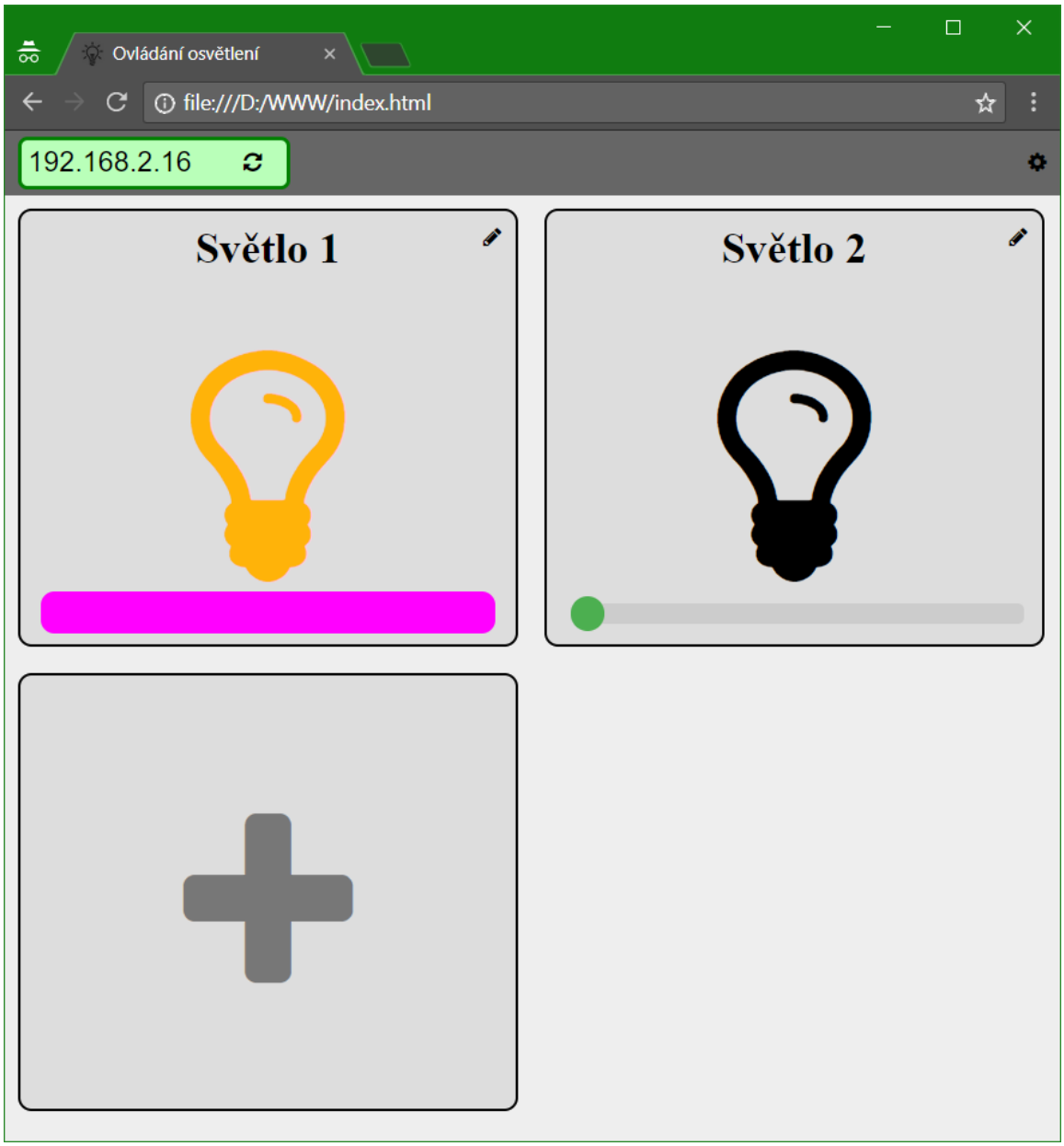
Obrázek A.3: Rozšířené zapojení



Obrázek A.4: Deska plošných spojů (*pohled z vrchu*)



Obrázek A.5: Deska plošných spojů (*pohled zespod*)



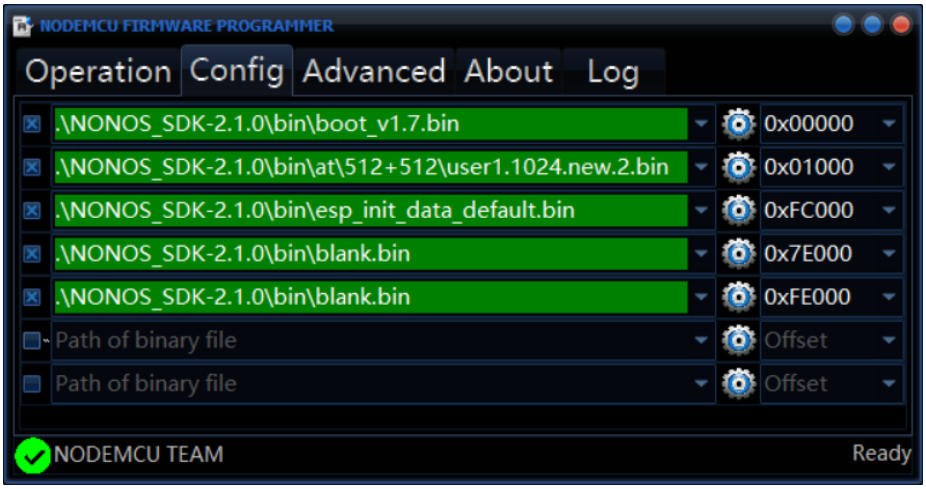
Obrázek A.6: Ukázka uživatelského prostředí na PC

Příloha B

Nahrání firmware do Wi-Fi modulu

Před nahráním je nutné propojit Wi-Fi modul s PC podle pokynů v odstavci 4.1.2. Dále je potřeba obstarat si program *NodeMCU Flasher* a firmware pro nahrání – já jsem zvolil *nonOS SDK* verze 2.1.0. Obojí se nachází na přiloženém CD v podsložce `ESP8266`.

V sekci *Config* se nastavují adresy pro nahrání jednotlivých částí firmware. Vhodné nastavení je vyobrazeno na obrázku B.1 (pozor na zaškrtávací políčka v levém sloupečku).

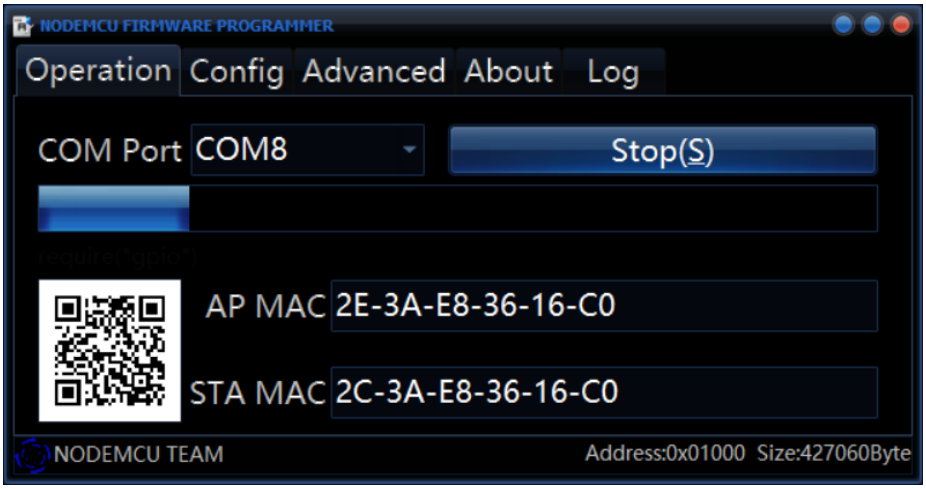


Obrázek B.1: Sekce *Config*

Poté je třeba v závislosti na použitém modulu nastavit sekci *Advanced*, v mém případě nastavení odpovídá obrázku B.2.

Obrázek B.2: Sekce *Advanced*

Na závěr je třeba v sekci *Operation* zvolil číslo COM portu, na kterém je připojený modul a až teď lze spustit nahrávání firmware.



Obrázek B.3: Sekce *Operation*

Příloha C

Technický popis komunikace

Komunikace mezi zařízením a uživatelem je velmi jednoduchá a obsahuje pouze 4 následující příkazy:

setLight<id>=<stav> nastaví světlo s indexem *id* na *stav*, přičemž stav může nabývat hodnot *on* pro zapnutí, *off* pro vypnutí, celého čísla v rozsahu 0 – 100 pro změnu jasu na daný počet procent nebo barvy ve tvaru #RRGGBB.

load<id>=<data> načte nové nastavení pro světlo s indexem *id*. Parametr *data* obsahuje binární data, která se přímo uloží do MCU bez toho, aby je bylo nutné zpracovat. Příkaz lze využít pro přidání, úpravu i smazání světla. *Data* jsou jeden byte, kde jednotlivé bity nesou následující informace `PPPPPTT1`:

- **P** značí číslo pinu.
- **T** představuje typ světla kódovaný následovně:

  00 značí odpojené světlo a slouží pro jeho smazání
  01 představuje normálně stmívatelné světlo
  10 je nestmívatelné světlo, tedy není validní operace pro změnu jasu
  11 popisuje RGB osvětlení
- **1** poslední bit je vždy 1 a zajišťuje, aby výsledné číslo nebylo nulové.

getStatus slouží pro získání aktuálního stavu všech připojených světlech. Odpovědí je pole bytů. Jejich počet je dvojnásobek maximálního počtu světel, které je zařízení schopné obsloužit. První byte nese informaci o prvním světle. Bity jsou uspořádány stejně jako v předchozím bodě s tím rozdílem, že poslední bit již nemusí být nutně `1`, ale značí, zda světlo svítí nebo ne (`1` znamená, že svítí). Druhý byte pak nese informaci o aktuálním jasu/barvě. Jas je uložen normálně jako číslo v rozsahu 0 – 100 a značí procenta. V případě barvy se jedná o 8-bitovou hloubku. Jak získat konkrétní barvu popisuji v sekci 4.4.3.

set<Client/Server>=<nastavení> nastaví Wi-Fi modul buď do režimu přístupový bod, tedy začne vysílat vlastní Wi-Fi síť (volba *Server*). Naopak v režimu *Client* se připojí do existující sítě.

Nastavení má tvar <ssid>&<heslo>&<kanál>&<šifrování>, kde:

- **SSID** nese informaci o názvu sítě. Maximálně 32 znaků bez diakritiky.
- **Heslo** je buď prázdné, pak se kóduje jako - nebo se jedná o 8 – 64 ASCII znaků.

- **Kanál** se používá pouze v režimu *Server* a jedná se o číslo v rozsahu 1 – 13.
- **Šifrování** je dostupné také pouze v režimu serveru a značí použitý typ zabezpečení sítě kódovaný jako číslo

  0 otevřená síť
  2 WPA_PSK
  3 WPA2_PSK
  4 WPA_WPA2_PSK

Příloha D

Obsah CD

Na přiloženém CD se nachází následující soubory a složky:

`Arduino` – veškeré zdrojové kódy pro MCU. Jejich členění zmiňuji dále.

`BP` – latexové soubory a obrázky použité pro vytvoření práce.

`Dokumentace` – generovaná nástrojem *Doxygen* ve tvaru webové stránky a pdf.

`DPS` – podklady pro výrobu DPS.

`ESP8266` – viz příloha B.

`Schémata` – elektronická schémata kreslená nástrojem Fritzing.

`WWW` – uživatelské rozhraní ve formě webové stránky.

`Zdroje` – uložené webové stránky ve formátu pdf, které sloužili jako podpůrná literatura.

`demo.mp4` – video-ukázka zařízení.

Členění zdrojových kódů

Kód jsem členil do logických bloků pomocí více souborů. Ve vyšších jazycích by každý soubor představoval jednu třídu a její metody. To jsem simuloval předponou, kterou jsem vložil na začátek všech funkcí v souboru, např. všechny funkce pro práci s Wi-Fi modulem se nacházejí v souborech `wifi.cpp` a `wifi.hpp` a jejich název vždy začíná předponou `wifi_` následovanou jménem funkce, přičemž jsou v rámci souboru seřazeny abecedně. Díky tomu lze snadno zjistit, ve kterém souboru se funkce nachází, což je vhodné pro hledání chyb a zároveň to zvyšuje čitelnost kódu.

Projekt jsem rozčlenil celkem do 17 souborů (z toho 10 hlavičkových). Vyskytují se v nich závislosti na standardní knihovny jazyka C (`string.h` a `math.h`) i Arduina (`EEPROM.h`, `SoftwareSerial.h` a `HardwareSerial.h`). Přípony zdrojových kódů `.cpp` a `.hpp` sice evokují použití C++, ale důvod je jiný. Jsou jím použité Arduino knihovny, které jsou psány v jazyce C++. Proto je nutné kód kompilovat překladačem pro C++, i když se jedná pouze o čisté C.

Kromě v části 4.2.3 probrané knihovny pro Wi-Fi modul a `main.cpp` souboru zmíněného v základní struktuře kódu (4.4.1) jsou zde následující:

availablePins.hpp deklaruje funkce pro správu výstupních pinů. Tedy funkce pro kontrolu, zda je daný pin možný použít tak, jak uživatel zamýšlí (stmívatelný nebo ne), jestli již není obsazený či zda vůbec existuje. Také zde probíhá překlad čísla pinu. Tudíž v rámci ovládání se používá pořadí pinu na svorkovnici namísto jeho skutečného umístění na MCU, což je pro uživatele mnohem přehlednější.

base64.hpp je knihovna pro zakódování dat pomocí Base64. Dekódování se v programu nepoužívá, proto nebylo implementováno.

color.hpp – důležitá knihovna pro práci s barvou. Barva je v rámci kódu reprezentována 3 způsoby. Ve všech třech případech se používá RGB model s rozdílnou bitovou hloubkou. Uživatelské rozhraní generuje barvy ve tvaru `#RRGGBB`. To znamená, že barva je popsána hexadecimálními čísly, přičemž se jedná o 24-bitovou barevnou hloubku. Tento tvar je pro strojové zpracování zcela nevhodný, proto je ihned přeložen do struktury `RGBColor`, která ukládá každou složku zvlášť jako jeden byte, takže se stále jedná o 24-bitovou hloubku (8 bitů na jednu barvu), ovšem oproti předchozímu tvaru postačí pro uložení 3 byty místo původních 7.

Z důvodu optimalizace (viz 4.4.4) je barva dále komprimována na 8 bitovou hloubku. Zde už dochází ke ztrátě informací. Uspořádání v rámci 8 bitů je `RRRGGGBB`. Pro (de)kódování se nepoužívá paleta, výsledné číslo je přímo vypočteno. Výpočet probíhá tak, že se ze vstupních barev vymaskuje pouze určitý počet nejdůležitějších bitů (pro červenou a zelenou jsou to tři, pro modrou pouze dva). Tyto bity jsou pak uloženy do jednoho byte, který reprezentuje výslednou barvu.

Převod opačným směrem probíhá obdobně. Extrahují se bity reprezentující danou barevnou složku a doplní se nulami. Pouze v případě, že se jedná o samé jedničky (tedy `111` pro červenou a zelenou nebo `11` pro modrou), doplní se jedničkami. To proto, aby bylo možné uchovat sytou barvu.

config.hpp protože jednotlivé knihovny obsahují různá makra (technicky vzato se jedná spíše o konstanty) ovlivňující jejich chování. Ty jsem vždy vložil na začátek hlavičkového souboru, aby bylo snadné v budoucnu modifikovat chování programu. Na druhou stranu, prohledávat všechny soubory a hledat konstanty není zrovna komfortní. Proto jsem vytvořil tento konfigurační soubor, který shlukuje všechny v programu použité konstanty do jednoho souboru.

To ale neznamená, že bych smazal hodnoty z hlavičkových souborů. Při kompilace pouze pomocí preprocesoru proběhne kontrola na existenci konstanty v konfiguračním souboru a pokud existuje, použije se její hodnota. V opačném případě se použije výchozí hodnota definovaná v hlavičkovém souboru.

debug.hpp obsahuje sadu ladících a informačních maker.

functions.hpp shromažďuje obecné funkce potřebné pro běh programu.

light.hpp obstarává práci se samotným osvětlením.

uart.hpp zapouzdře komunikaci po sériové lince. Hlavní funkcí je odstínit zbytek programu od detailů, tedy zda se jedná o hardwarový či softwarový port. Snadno tak lze úpravou jedné funkce převést program na jiný MCU, který obsahuje více UART modulů.